18ème Atelier "Raisonnement à Partir de Cas" RàPC 2010

Florence Le Ber, Jean Renaud

18ème Atelier « Raisonnement à Partir de Cas »

RàPC 2010

Strasbourg, 29-30 juin 2010

Florence Le Ber, Jean Renaud, éditeurs

Préambule

Le raisonnement à partir de cas (RàPC) est un paradigme de résolution de problèmes s'appuyant sur la réutilisation d'expériences passées pour résoudre de nouveaux problèmes. Les applications du RàPC sont nombreuses et la recherche est particulièrement active en France et dans le monde. Les rencontres annuelles de la communauté française ont été organisées depuis 1992 par le groupe français de recherche en RàPC[1], sous la forme d'ateliers d'un à deux jours, permettant de présenter et de discuter les travaux, théoriques ou appliqués, à différents stades d'avancement.

Cette année 2010, le 18ème atelier RàPC est organisé à Strasbourg, en amont des assises du GDR I3 (« Information, Interaction, Intelligence »). À cette occasion, l'atelier RàPC partage une demi-journée avec les rencontres du thème IAF « Intelligence Artificielle Fondamentale » du GDR I3. Le programme complet est ainsi constitué de neuf présentations, huit soumises à l'atelier RàPC et une soumise aux journées IAF. Ces présentations sont réparties en quatre sessions : une première session porte sur des applications du RàPC à l'espace et aux déplacements ; une deuxième session (en deux temps) regroupe différents travaux sur l'adaptation ; les deux autres sessions sont consacrées pour l'une à la réutilisation d'expériences et à la remémoration, et pour l'autre à la comparaison du RàPC à d'autres méthodes appuyées sur l'expérience. De plus, l'atelier accueille deux exposés-tutoriaux, l'un sur la méthode TRIZ, par Roland de Guio, responsable du LGECO, à l'INSA Strasbourg, l'autre sur les liens entre extraction de connaissances et raisonnement à partir de cas, par Amedeo Napoli, responsable de l'équipe Orpailleur du LORIA Nancy.

Les deux responsables scientifiques de cet atelier 2010 remercient les auteurs pour leurs contributions ainsi que les lecteurs qui ont effectué un travail soigné et constructif, afin d'offrir à tous un atelier de qualité. Nous remercions également l'INSA Strasbourg qui accueille l'atelier le mardi 29 juin, ainsi que les organisateurs des assises du GDR I3, en particulier Pierre Gançarski, et les responsables des journées IAF, Laurence Cholvy et Sébastien Konieczny, pour l'organisation de la demi-journée conjointe, le 30 juin.

Florence Le Ber et Jean Renaud

[1] `http://liris.cnrs.fr/~rapc/mediawiki/index.php/Accueil`

Comités

Comité de programme

- Fadi Badra (LORIA, Nancy),
- Mathieu d'Aquin (Kmi, Milton Keynes, UK),
- Amélie Cordier (LIRIS, Lyon),
- Pierre De Loor (LISyC, Brest),
- Sylvie Després (LIPN, Paris),
- Béatrice Fuchs (LIRIS, Lyon),
- Marie-Christine Jaulent (SPIM, Paris),
- Rushed Kanawati (LIPN, Paris),
- Florence Le Ber (LHyGeS, Strasbourg & LORIA, Nancy),
- Jean Lieber (LORIA, Nancy),
- Sophie Loriette-Rougegrez (Institut Charles Delaunay, Troyes),
- Alain Mille (LIRIS, Lyon),
- Brigitte Morello (FEMTO-ST, Besançon),
- Amedeo Napoli (LORIA, Nancy),
- Jean Renaud (LGECO, Strasbourg),
- François Rousselot (LGECO, Strasbourg),
- Sylvie Salotti (LIPN, Paris),
- Karim Sehaba (LIRIS, Lyon)

Comité d'organisation

- Florence Le Ber (ENGEES Strasbourg),
- Jean Renaud (INSA Strasbourg),
- François Rousselot (INSA Strasbourg)

Programme de l'atelier

Mardi 29 à l'INSA

Accueil à partir de 9h

9h45-10h45 Session "Espace et déplacements"

Raisonnement à partir de scénarios pour l'assistance au déplacement – *Baptiste Cablé, Sophie Loriette, Jean-Marc Nigro, Yann Barloy*

Cas socio-spatiaux pour le diagnostic prospectif de territoires – *Sylvie Lardon, Florence Le Ber*

10h45-11h15 Session "Adaptation (1)"

Un algorithme d'adaptation avec des cas exprimés dans la logique de descriptions ALC – *Julien Cojan, Jean Lieber*

11h15-11h45 Pause

11h45-13h Session "TRIZ"

Comment construire des méthodes de résolution de problèmes d'invention : propositions de la TRIZ – *Roland de Guio*

Comparaison d'une méthode de Conception inventive basée sur la méthode TRIZ et de l'approche du raisonnement à partir de cas – *Denis Cavallucci, François Rousselot, Jean Renaud*

13h-14h30 Repas

14h30-16h Session "Remémoration et réutilisation"

Améliorer la remémoration par enrichissement de l'ontologie du domaine – *Valmi Dufour-Lussier, Jean Lieber, Emmanuel Nauer, Yannick Toussaint*

Providing assistance by reusing episodes stored in traces: a case study with SAP Business Objects Explorer – *Raafat Zarka, Amélie Cordier, Françoise Corvaisier, Alain Mille*

Raisonnement à partir de cas dynamique pour la réutilisation contextuelle de l'expérience – *Amélie Cordier, Bruno Mascret, Alain Mille*

16h-16h30 Pause

16h30-17h30 Discussion

Mercredi 30 au pôle API : Journée commune avec le thème IAF du GDR I3

9h30-11h : Tutoriel

Quelques apports de l'extraction de connaissances au raisonnement à partir de cas – *Amedeo Napoli*

11h-11h30 Pause

11h30-12h30 Session "Adaptation (2)"

Analyse formelle de concepts pour l'adaptation de cas textuels – *Valmi Dufour-Lussier, Jean Lieber, Emmanuel Nauer, Yannick Toussaint*

Adapter des cas en utilisant un opérateur de révision ou des règles – *Julien Cojan, Jean Lieber*

Table des matières

Raisonnement à partir de scénarios pour l'assistance au déplacement

Baptiste Cablé, Sophie Loriette, Jean-Marc Nigro, Yann Barloy,
Institut Charles Delaunay, Université de technologie de Troyes
12 rue Marie Curie, B.P. 2060, 10 010 Troyes Cedex
Email : prenom.nom@utt.fr

22 mars 2010

Résumé

Les travaux présentés s'inscrivent dans le cadre de l'assistance à la conduite en fauteuil roulant. Le projet VAHM (véhicule autonome pour handicapé moteur) a proposé, il y a quelques années, une solution semi-automatisée pour commander un fauteuil roulant à l'aide d'un seul bouton. Le fauteuil est équipé de capteurs lui permettant d'éviter les obstacles. Cependant, à chaque changement de direction, l'utilisateur doit patienter pour activer le bouton à un moment précis, ce qui est contraignant. Nous détaillerons une technique d'intelligence artificielle pour mémoriser les déplacements effectués de manière à proposer automatiquement, à partir de l'expérience acquise, la direction à venir. Notre solution, inspirée du raisonnement à partir de cas, s'adapte aux évolutions de l'environnement ainsi qu'aux changements d'habitudes de l'utilisateur. Ce papier présente l'aspect théorique de la solution puis décrit les expérimentations et les résultats.

Mots clés : reconnaissance de scénarios, anticipation, situations dynamiques, apprentissage symbolique.

1 Introduction

Nos recherches s'inscrivent dans le cadre de l'assistance au déplacement des personnes fortement handicapées. Le projet VAHM propose un fauteuil qui nécessite de n'agir que sur un seul bouton pour se diriger. Cependant, parcourir un trajet usuel peut amener l'utilisateur à entrer un grand nombre de commandes, et ce de manière précise. Cette démarche étant contraignante, nous proposons d'intégrer au système une méthode d'intelligence artificielle qui assiste l'utilisateur dans le déplacement du fauteuil en prenant, à sa place, les décisions les plus probables. Ainsi, l'utilisateur n'intervient que si l'application a mal prédit la direction à suivre.

Cette solution s'appuie sur une méthode de raisonnement à partir de scénario que nous avons conçue en nous inspirant du raisonnement à partir de cas. Elle construit, sous forme d'un ensemble de scénarios, un modèle du système en mémorisant les scénarios. Son but est ensuite de se servir de ces connaissances pour prédire l'action à effectuer en fonction de l'état observé et du début du scénario.

La section 2 est consacrée à une présentation du contexte de la reconnaissance et l'anticipation de situations dynamiques ainsi qu'à une introduction au projet VAHM. La section 3 décrit la solution proposée pour anticiper les situations dynamiques. La section 4 présente l'application de notre solution au fauteuil VAHM et commente les résultats obtenus. Nous terminons par une conclusion débouchant sur les perspectives de travaux futurs.

2 Contexte

2.1 Représentation et anticipation de scénarios

Plusieurs outils sont couramment utilisés pour décrire et manipuler les scénarios. Parmi les plus utilisés en modélisation, on trouve les réseaux de Petri (RdP) [8]. Un RdP est un graphe orienté comprenant un ensemble de places et de transitions reliées par des arcs. Le déplacement de jetons de places en places modélise l'évolution de l'état du système. Le RdP est un outil mathématique très précieux pour modéliser visuellement le déroulement

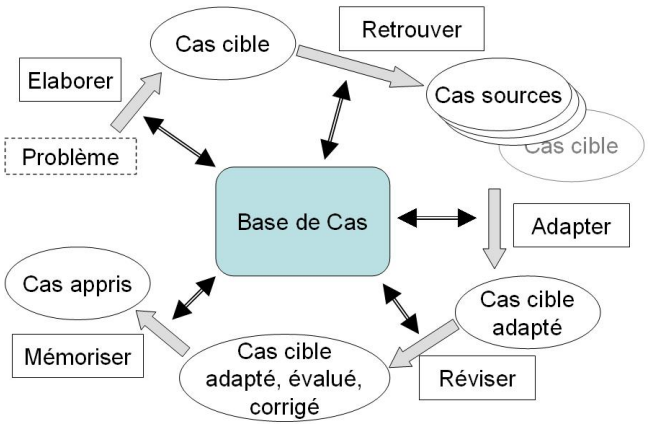


FIGURE 1 – Raisonnement à partir de cas

d'un scénario, cependant il n'a pas été conçu dans le but d'anticiper. De plus, l'apprentissage et la mise à jour des RdP sont complexes. Nous n'utiliserons pas cet outil dans nos travaux.

Les réseaux bayésiens dynamiques (RBD) sont une extension des réseaux bayésiens statiques [3]. Ils permettent de modéliser des distributions de probabilité sur des ensembles semi-infinis de variables aléatoires (Ut, Xt, Yt) représentant les variables d'entrée, les états cachés et les sorties d'un modèle d'état [4]. Les RBDs généralisent les filtres de Kalman et les chaînes de Markov (cachées et hiérarchiques). Leur utilisation permet de faire de la surveillance et de la prédiction de l'état futur du système dynamique. Leur utilisation est cependant complexe et nécessite beaucoup de connaissances de la part du concepteur. De fait, ils ne peuvent répondre à notre problématique car nous voulons une solution légère (en termes de rapidité) et qui ne demande presque aucune connaissance au concepteur quant aux déroulements possibles du système.

Un modèle de Markov caché [2] est constitué d'une variable cachée ou interne représentant l'état du système à modéliser. Cet état interne n'est pas observable directement. En revanche, une autre variable, conséquence de cet état interne, est observable. Les modèles de Markov cachés sont inclus dans les RBDs mais il faut bien noter que leur aspect "caché" ne nous intéresse pas, ici.

Le raisonnement à partir de cas (RàpC, figure 1) est une méthode d'intelligence artificielle dont l'idée générale est de chercher à résoudre un problème en adaptant des solutions de problèmes résolus précédemment [5]. Ce raisonnement est basé sur l'expérience.

Parmi ces méthodes, celle qui nous intéresse et a inspiré nos travaux et le RàpC. Citons par exemple dans ce domaine les travaux d'Alain Mille à propos de raisonnement à partir de l'expérience tracée [6], où les traces d'interaction sont réutilisées pour résoudre un problème.

2.2 VAHM

Le projet VAHM (Véhicule autonome pour Handicapés Moteurs) [9] a pour but de proposer aux personnes ayant un handicap physique lourd de se déplacer facilement. L'interface est composée d'un bouton et d'un cadrant indiquant successivement huit directions de déplacement. Lorsque l'utilisateur veut suivre une direction, il doit attendre qu'elle s'affiche sur le cadran et presser le bouton à ce moment. Lorsqu'il relâche le bouton, le fauteuil s'arrête et il faut choisir une nouvelle direction. Ce système présente l'avantage de ne nécessiter qu'un seul type d'action de la part de l'utilisateur, mais a l'inconvénient d'être lent et fastidieux. C'est pourquoi le fauteuil doit se déplacer de la manière la plus autonome possible. Pour cela, il est équipé de capteurs qui collectent des données sur son environnement et d'agents réactifs qui lui permettent des comportements simples (longer les murs, éviter les obstacles, etc.). Lors de ce projet, un simulateur a été développé. Celui-ci permet de remplacer les expérimentations avec un véritable fauteuil roulant par des expérimentations informatiques, plus rapides, donnant des résultats sensiblement proches.

L'objectif de notre recherche est de doter le fauteuil d'un système qui reconnait les différents trajets effectués régulièrement de manière à reproduire automatiquement la fin du trajet, mais sans avoir recours à la

cartographie. Différentes versions de ce système ont déjà été conçues : dans l'article [7], une approche à base de métaconnaissance et de système à base de règles est proposée. En considérant la succession des comportements simples du fauteuil, le système crée des classes de trajets. Lors d'un nouveau trajet, il détermine si ce trajet correspond à une classe existante ou s'il est nécessaire de créer une nouvelle classe (ou d'en modifier une existante). Lors de sa thèse, A. Aich étudie deux autres approches. La première s'appuie sur les réseaux de Petri mais ne donne pas de résultats suffisants. En effet, l'apprentissage est complexe et les données brutes étudiées sont insuffisantes. La deuxième, SAPED [1], basée sur le raisonnement à partir de cas, est plus performante et permet un meilleur apprentissage des nouveaux trajets. Le travail est cette fois-ci appliqué à des segments de trajets qui sont mis bout-à-bout pour créer un trajet. Bien que des progrès importants soient réalisés en terme de reconnaissance de trajet, cette solution ne permet qu'une autonomie du fauteuil en fin de trajet (le début est consacré à la reconnaissance).

3 Anticipation à partir de scénarios

Cette section présente le principe de la méthode de raisonnement à partir de scénarios que nous avons imaginé. Quelque soit le domaine d'application (VAHM ou autre) notre solution assiste l'utilisateur. L'application est capable de déclencher des actions mais l'utilisateur peut le contredire et reste maître du système.

3.1 Vocabulaire

Un *état* est une situation statique dans laquelle se trouve le système à un instant ponctuel. Pour une voiture, un état pourrait par exemple être "dépassement en cours" et pour une porte automatique cela pourrait être "fermée".

Une *action* correspond à un évènement concernant le système et qui modifie généralement son état. Exemples : pour une voiture, "tourner à droite", et pour une porte automatique, "ouverture".

Un *scénario* est une séquence ordonnée de couples (etat, action) du système. Pour une porte automatique, cela pourrait être "(passant détecté, ouverture de la porte) ; (porte ouverte et présence détectée, maintenir la porte ouverte) ; (porte ouverte et plus de présence, fermer la porte)". Au travers de cet exemple, on s'aperçoit qu'avec ce type de représentation, certains couples (état, action) ne sont pas indispensables au bon déroulement du scénario mais ils permettent seulement de le reconnaître avec plus de certitude. Il s'agit ici du deuxième couple, où aucune action n'est réellement effectuée. Il permet tout de même de différencier ce scénario de celui où personne ne traverse la porte. De tels couples sont dits *passifs* : l'action consiste en "aucune action". Par opposition, on appelle couples *actifs* tous les autres couples, c'est à dire ceux où une action a réellement lieu et qui sont indispensables à la reproduction du scénario.

Le *scénario courant* (SC) est le scénario observable qui est en train de se dérouler.

Un *scénario source* (SS) est un scénario connu de notre système dont le début correspond au début observé du scénario courant. Il peut y avoir plusieurs scénarios source à la fois.

Le *score* d'un scénario est une grandeur qui donne une indication sur sa probabilité d'être reproduit. Plus un scénario est souvent et récemment reproduit, plus son score est élevé. Attention, le score n'est pas une valeur de probabilité ; c'est une grandeur qui, une fois comparée aux scores des autres scénarios, permet de déterminer le plus probable. Le score est utilisé lors de la prédiction pour déterminer la direction à suivre la plus probable. Il est mis à jour à la fin du déroulement de chaque scénario. Au moment du choix de la prédiction, le score d'une action est calculé. Il découle directement des scores des scénarios proposant cette action : c'est la somme de ces scores.

3.2 L'algorithme de raisonnement à partir de scénarios

L'algorithme crée un modèle du système sur lequel il s'appuie pour raisonner. L'objectif du modèle est de représenter l'ensemble des scénarios de déroulement possibles du système afin d'anticiper les actions à venir. Toutes les actions possibles sont connues. Cependant, les scénarios et les actions sont initialement inconnus ; ils sont appris en ligne. L'application modélise donc le système par :

– l'ensemble des états, $\mathcal{E}$, non exhaustif, initialement vide.

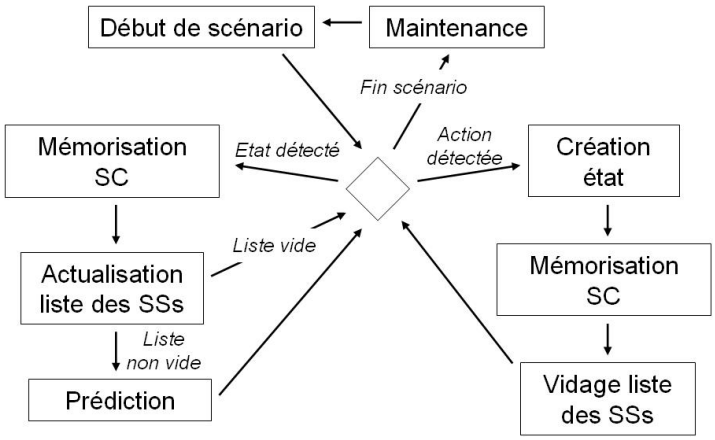


FIGURE 2 – L'algorithme de raisonnement à partir de scénarios

– l'ensemble des actions, $\mathcal{A}$, fixé.
– l'ensemble des scénarios, $\mathcal{S}$, non exhaustif, initialement vide.

Les états e_i de $\mathcal{E}$ ainsi que les actions a_i de $\mathcal{A}$ sont des symboles. Un scénario de longueur n est un n-uplet de couples (e_i, a_i). Par exemple, le scénario s_4, de longueur 8, est de la forme : $s_4 = ((e_{4,1}, a_{4,1}), (e_{4,2}, a_{4,2}), ..., (e_{4,8}, a_{4,8}))$.

Le fonctionnement de l'algorithme est résumé sur la figure 2. Lorsqu'un nouveau SC débute et qu'aucun état n'a été observé, tous les scénarios de S sont des SSs car leur début correspond au début du SC. Ils sont donc tous ajoutés à $\mathcal{L}$, la liste des scénarios source. Dés l'observation du premier état et ensuite à chaque nouvel état (côté gauche sur le schéma), chaque scénario de $\mathcal{L}$ est contrôlé pour vérifier qu'il possède le dernier couple (e_i, a_i) observé du SC. Si ce n'est pas le cas, ce SS est exclu de $\mathcal{L}$.

A chaque nouvel état, avant que l'action ne soit déclenchée par l'utilisateur, l'anticipation consiste à prédire, comme action à effectuer, celle qui est la plus probable. Pour ce faire, un score est attribué à chaque action possible en fonction des scores des SSs correspondant. Si une action est prédite, elle est exécutée à moins d'être contredite par l'utilisateur.

Comme tous les états ne sont pas initialement connus, il peut arriver qu'une action soit déclenchée sans que l'état associé n'ait été reconnu (car non appris). Cette possibilité est envisagée sur la partie droite du schéma : une action est détectée. A ce moment, l'état associé à l'action est appris pour que le scénario puisse être reconnu par la suite. Le scénario est forcément nouveau sinon l'état en question aurait déjà été appris car indispensable à la description du scénario (couple actif, cf. 3.1). La liste des scénarios source est donc vidée. Plus aucune anticipation ne sera faite jusqu'à la fin du scénario.

3.3 Les scores

A la fin du scénario, la phase de maintenance se déroule. Son but est d'assurer la cohérence des connaissances du système. Lors de cette phase, les scores sont mis à jours.

3.3.1 Evolution des scores

La prédiction de l'action à effectuer est une des finalités de notre algorithme. Le calcul du score des actions possibles est donc un point clé. Le score d'une action possible est la somme des scores de tous les SSs proposant cette action. Il faut donc que le score de chaque SS reflète la probabilité du scénario d'être reproduit. Nous avons choisi de faire évoluer les scores des scénarios de manière à ce que deux éléments soient pris en compte : le nombre de reproductions du scénario et leur caractère récent. De plus, ce score doit pouvoir tomber à zéro, ce qui correspond à un oubli du scénario car il n'est pas suffisamment utilisé. Pour terminer, il faut que sommer plusieurs scores donne un résultat reflétant la probabilité de l'union de ces scénarios. Une loi mathématique convenable, pour tenir compte de ces éléments, est la suivante :

– à chaque utilisation d'un scénario, son score est augmenté de manière affine.

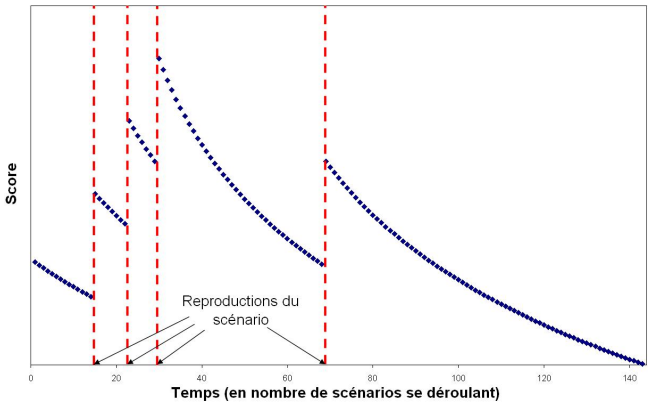


FIGURE 3 – Evolution du score d'un scénario

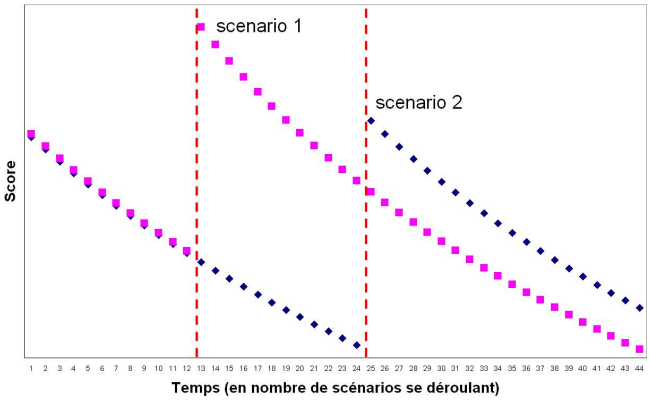


FIGURE 4 – Evolution de deux scores initialement identiques de scénarios

– à chaque non-utilisation, son score est diminué de manière logarithmique (évolution en $-\ln(x)$).

3.3.2 Interprétation de la valeur des scores

Un exemple de loi de ce type est présenté sur la figure 3. Ce graphique indique la valeur du score d'un scénario en fonction du temps. Le temps est compté en nombre de scénarios se déroulant, ce qui signifie que si le fauteuil reste au repos, les scores n'évoluent pas. Le score est augmenté à chaque fois que le scénario en question se déroule (ligne pointillées sur le graphique) et il est diminué lorsque tout autre scénario se déroule. La courbe obtenue confirme que plus le score d'un scénario est faible, plus son score décroit lentement, mais que quelque soit son score, une utilisation le fait toujours autant augmenter. Ainsi, comme souhaité, plus un scénario est reproduit ou plus son utilisation est récente, plus son score est grand. Le graphique de la figure 4 compare l'évolution du score de deux scénarios dont le score est initialement égal. Le scénario 1 est reproduit le premier et son score devient supérieur à celui du scénario 2 aussi longtemps qu'aucun des deux ne sera utilisé. Cependant, de fait, il décroit plus rapidement. Ainsi, quand le scénario 2 est à son tour reproduit, son score augmente et devient le plus important. Nous avons là le résultat souhaité : pour deux scores initialement identiques, pour une reproduction de chacun des scénarios, celui qui a été reproduit le plus récemment obtient un meilleur score.

La loi de croissance, suite arithmétique, est de la forme $u_{n+1} = u_n + a$, avec a, réel, le paramètre qui détermine la vitesse de croissance. La loi de décroissance, logarithmique, est de la forme $v_{n+1} = \log_{(1-a)}\left[(1-a)^{v_n} + b\right]$, où b est le paramètre qui détermine la vitesse de décroissance. Lorsque le score d'un scénario tombe à zéro car il n'est pas suffisamment souvent utilisé, il est enlevé de $\mathcal{S}$ (l'ensemble des scénarios du système). Il est comme "oublié" par le système, ce qui permet d'une part d'éviter de proposer des prédictions obsolètes et d'autre part

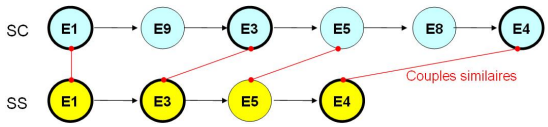


FIGURE 5 – Décalage entre le SC et un SS

de renouveler automatiquement le modèle du système lorsque les scénarios habituellement effectués changent.

3.4 Décalage

La conséquence de l'ajout d'états est que certains SS ne correspondent plus au SC alors qu'en réalité, le même scénario de déroule. En effet, reprenons l'exemple de la porte automatique (cf. 3.1) et supposons qu'il s'agisse le début de l'exécution de l'algorithme. L'action "ouverture de la porte" est détectée. L'état associé est donc appris. Lorsqu'ensuite la porte est ouverte et que le passant la traverse, il n'y a pas d'action de la part du système. L'état n'est donc pas appris. A la fin du scénario, l'algorithme aura appris "(passant détecté, ouverture de la porte) ; (porte ouverte et plus de présence, fermer la porte)" et effectivement, cela suffit à reproduire le scénario par la suite uniquement grâce à l'observation des états. Si plus tard, pour un autre scénario, l'état "porte ouverte et présence détectée" est associé à une action et donc appris, lorsque notre premier scénario (celui à 2 couples) se déroule de nouveau, le SS ne correspond pas au SC. En effet, le deuxième état du SC est "porte ouverte et présence détectée" (qui peut désormais être reconnu) alors que le SS attend d'observer "porte ouverte et plus de présence". Pourtant, le scénario réel est exactement le même. Cela est problématique pour notre système car le SS serait exclu à tord de $\mathcal{L}$ et ne participerait donc pas à la prédiction alors qu'il est le bon scénario.

En résumé, des couples passifs peuvent manquer dans les SSs pour correspondre aux SCs, bien que ces SSs soient justes. Pour remédier à ce problème, un décalage (fig. 5) est toléré entre les SSs et le SC. Un SS auquel il ne manque que des couples passifs ne peut pas être supprimé de $\mathcal{L}$. Ainsi, il participe toujours aux prédictions.

3.5 Nouveaux scénarios

A la fin de chaque scénario, lors de la maintenance, les scores sont mis à jour et le SC appris, s'il est nouveau. S'il reste un SS à la fin du scénario, cela signifie qu'il correspond au SC. Cependant, il peut y avoir un certain décalage entre les deux (couples passifs manquants). Si c'est le cas, son score n'est pas augmenté et le nouveau scénario est appris. Ainsi, le décalage permet aux SS qui ne sont pas à jour de participer à la prédiction, mais sans considérer qu'ils sont exactement corrects.

Plutôt que de mettre à jour un ancien scénario valide, le choix est donc fait d'en apprendre un nouveau. Durant une période de transition, deux scénarios de $\mathcal{S}$ correspondent au même déroulement du système et les deux influeront dans le même sens pour la prédiction. Cependant, celui qui est à jour verra son score augmenter à chaque reproduction tandis que le score de l'ancien diminuera constamment. Durant cette période, la somme des scores sera quasiment égale au score qu'aurait eu le scénario ancien s'il avait été mis à jour, du fait des lois mathématiques choisies. Après un moment, seul celui qui est à jour perdurera. L'intérêt de cette méthode est multiple. Tout d'abord, il est plus simple d'enregistrer un second scénario que de chercher à en mettre un jour. Ensuite, cela évite d'éventuelles mises à jour abusives ; un même scénario peut correspondre à plusieurs situations dynamiques légèrement différentes ; elles peuvent être différentiées en ajoutant des couples passifs dans les moments caractéristiques, si bien qu'ajouter le couple au scénario peut être vrai pour une des situations mais faux pour les autres. En créant un nouveau scénario, l'ancien perdure et peut toujours être activé. Dans le cas d'un scénario qui correspondait à plusieurs situations dynamiques, l'une d'entre elles se retrouve précisée par le nouveau scénario mais les anciennes sont toujours représentées par l'ancien (fig. 6).

Dans cet exemple, s_1 correspond aux trois scénarios (A, B et C). Si l'état $E9$ est appris et que le scénario A se déroule à nouveau, s_1 participera aux prédictions mais il aura à la fin un décalage (d'un couple). Aucun SS existant ne restera en fin de scénario et s_2 sera donc appris. Cependant, s_2 ne correspond pas à B ni à C. Il est donc justifié de conserver s_1.

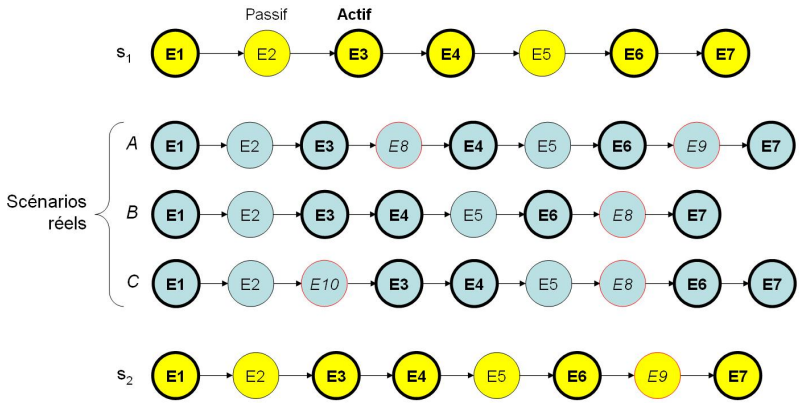


FIGURE 6 – Plusieurs scénarios courants pouvant être associés au même scénario source (s_1).

3.6 En termes de raisonnement à partir de cas

D'un point de vue global, notre algorithme s'apparente à du raisonnement à partir de cas. Un scénario est un cas. La phase d'élaboration correspond ici à la mémorisation (en temps réel) du cas sous la forme d'un enchaînement d'états et d'actions. Le problème cible est donc ce que nous appelons scénario courant. L'étape de remémoration se fait en continu et en temps réel dans notre algorithme : il s'agit de l'élimination des scénarios cible de la liste qui ne correspondent plus au scénario courant. Les cas cibles sont donc nos scénarios cible.

Là où notre algorithme diffère du raisonnement à partir de cas, c'est que rien ne correspond véritablement aux phases d'adaptation et de révision. En effet, dès la connaissance des cas source (et en permanence), une "solution" est proposée qui correspond à l'action la plus probable à effectuer. Il n'y a aucune adaptation des connaissances apprises, ni pour la résolution, ni pour la mémorisation. Il en découle que notre méthode permet uniquement de reproduire des scénarios déjà vus et non d'anticiper lors d'une situation nouvelle. Cependant, ceci n'est pas un inconvénient mais une volonté affirmée dès le début de la réalisation de l'algorithme au vu du domaine d'application souhaité. Une petite souplesse est d'ailleurs donnée par le choix de la méthode de représentation des états. En effet, des états permettant pouvant être reconnus avec une large tolérance permettra d'anticiper dans des situations physiquement différentes, mais sémantiquement identiques (si les états sont bien étudiés).

4 Application à VAHM

Afin de vérifier son efficacité, l'algorithme a été appliqué au projet VAHM (cf. 2.2), en prenant pour scénarios les trajets effectués. Le fauteuil dans son environnement est un système qui satisfait les conditions nécessaires pour se voir appliquer notre algorithme : ce système peut être discrétisé, les scénarios sont censés se reproduire régulièrement, la fin d'un scénario dépend du début et une assistance peut être proposée à l'utilisateur en pilotant automatiquement le fauteuil.

Dans cette application, un état correspond à une situation statique du système. Il décrit l'endroit auquel se trouve le fauteuil. Une action correspond à une direction à prendre (ou à un arrêt). L'action est habituellement effectuée par l'utilisateur. Le scénario est un trajet, qui se trouve donc décrit par une succession de couples (endroit, action). C'est d'ailleurs la méthode que nous, humains, utilisons pour expliquer à une personne comment trouver son chemin : "En partant de l'endroit X, tu vas tout droit. Quand tu auras un couloir sur ta droite, tu continues tout droit, puis deuxième porte à gauche.", ce qui sous-entend qu'à la première, il ne faut rien faire (couple passif).

Grâce à notre algorithme, le fauteuil peut prédire la direction qu'il va falloir choisir et l'utilisateur n'intervient que si cette prédiction est mauvaise. Sinon, le fauteuil se déplace de manière autonome. De fait, l'utilisateur aura beaucoup moins d'actions à effectuer, ce qui est très intéressant car ces actions lui sont coûteuses (cf. 2.2).

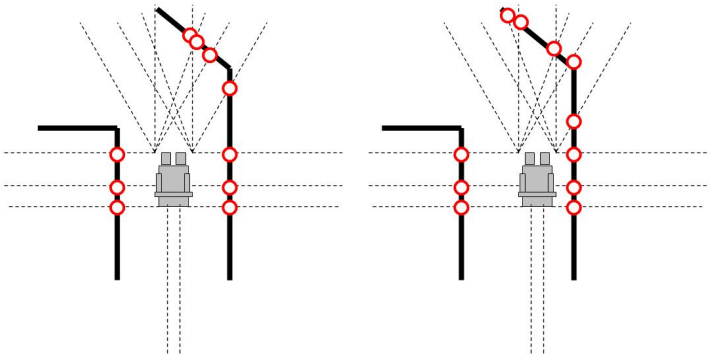


FIGURE 7 – Deux états équivalents

4.1 Approches testées

La difficulté de l'adaptation de notre algorithme à une application réside dans la représentation et la reconnaissance des états. En effet, c'est le seul élément qui soit dépendant de l'application et il demande l'intervention d'un expert. Il faut trouver une structure qui permette de mémoriser les états mais surtout de déterminer efficacement si le système se trouve dans un état connu, et si oui, lequel.

Pour le projet VAHM, cela a consisté à trouver comment représenter efficacement l'environnement proche du fauteuil. Une description et une reconnaissance trop précises sont mauvaises car l'état ne se reproduira presque jamais à l'identique. Les états ne seront donc jamais considérés reconnus et les scénarios se multiplieront sans être jamais reproduits. Inversement, des états trop vagues seront reconnus n'importe quand, ce qui rend impossible le bon fonctionnement du système.

Notre première idée, la plus intuitive, a été de représenter un état par un 16-uplet dont chaque valeur est la grandeur retournée par un des capteurs ultrasons entourant le fauteuil (le fauteuil est équipé de 16 capteurs de distance qui sont son seul moyen de percevoir l'environnement). Cependant, pour estimer si lors d'un état, on peut considérer qu'un état proche (déjà appris) est reconnu, il faut calculer la distance entre les deux 16-uplets. Quelle distance mathématique utiliser ? Celles qui donnent les moins mauvais résultats sont la distance de Tchebychev (∞-distance) et la distance euclidienne, mais des états peuvent tout de même être reconnus alors qu'ils sonts très différents, ou inversement. Sur l'exemple d'état de la figure 7, les deux états seraient considérés différents car les valeurs des capteurs pour l'état de droite seraient à droite du fauteuil trop faibles et à gauche trop élevées. Inversement, dans le cas de la figure 8, les deux seraient considérés identiques car les valeurs des capteurs sont sensiblement proches.

Nous avons donc envisagé d'autres approches, toujours basées sur des n-uplets, mais quelque soit leur dimension et la manière dont ils sont calculés, la reconnaissance des états est médiocre.

Nous avons ensuite développé une méthode géométrique de description d'états, comme une cartographie très locale. Cependant, la reconnaissance d'états nécessite de comparer les états appris avec l'état courant à chaque instant, ce qui est coûteux en calculs et donc contraire à la volonté de simplicité donnée dans notre problématique.

La solution finalement choisie est à base de règles. Un état est représenté par un ensemble de règles simples dont les conditions portent sur les grandeurs des capteurs. Lors de la création de l'état, les règles sont créées aléatoirement selon des modèles prédéfini, de manière à décrire l'endroit où se trouve le fauteuil. Ces règles sont essentiellement basées sur des comparaisons entre des combinaisons de valeurs de capteurs entre elles, si bien qu'une translation du fauteuil de quelques centimètres, ou un changement d'un détail dans l'environnement ne va pas empêcher la reconnaissance de l'état. Cette dernière consiste à valider un nombre suffisant de ces règles pour considérer que l'état est reconnu. Les règles visent, entre autre, à déterminer l'orientation des murs, la convexité de la pièce, etc. Ainsi, les deux exemples (fig. 7 et fig. 8) ne posent plus de problèmes.

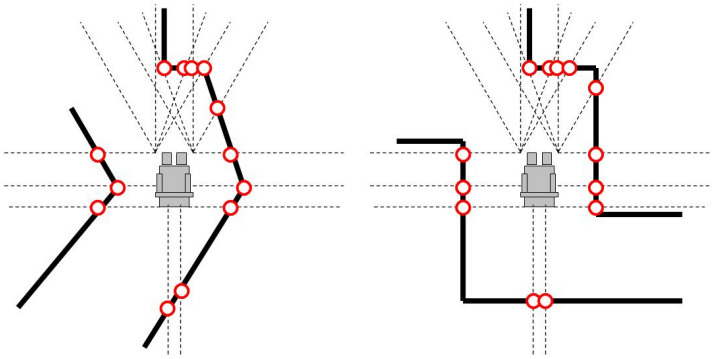


FIGURE 8 – Deux états différents

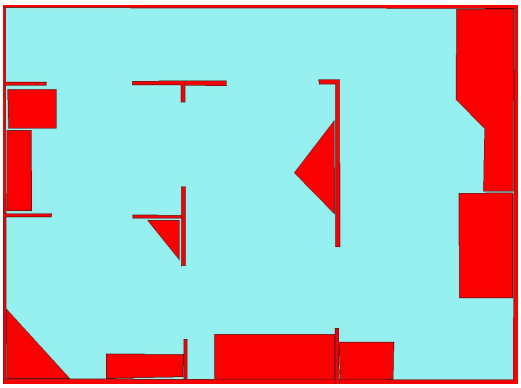


FIGURE 9 – Environnement d'expérimentation

4.2 Résultats expérimentaux

Les résultats obtenus lors des expérimentations avec le simulateur sont globalement bons, mais sont très difficilement quantifiables. La réduction du nombre de commandes à saisir par l'utilisateur dépend de nombreux paramètres : aspect et dimensions du lieu de vie, bonne utilisation de l'outil, fréquence de reproduction des trajets, nombre de trajets différents, etc. Afin de pouvoir donner des résultats numériques, nous avons fait une série d'expérimentations dans l'environnement présenté sur la figure 9.

Pour cette expérimentation, nous faisons une utilisation "normale" du fauteuil en reproduisant les trajets qui pourraient réellement être faits par un utilisateur sur une période de plusieurs jours. L'environnement n'évolue pas mais les habitudes de l'utilisateur ne sont pas forcément constantes. Chaque journée n'est pas identique. Bien qu'aucun trajet ni aucun état ne soit connu au départ, nous ne cherchons pas spécialement à éduquer le fauteuil mais simplement à faire les trajets comme dans la vie quotidienne. Le nombre de trajets différents envisagés ici est de 15. Les paramètres a et b des lois de croissance et décroissance sont respectivement fixés à $0,5$ et $0,02$. Ces valeurs traduisent une adaptation plutôt rapide du système à l'environnement. Un trajet parcouru une seule fois est oublié au bout de 25 trajets. Nous étudions la proportion de commandes à entrer par l'utilisateur par rapport au nombre total de commandes (manuelles + automatiques) en fonction du nombre de trajets.

Les résultats obtenus (fig. 10) confirment l'efficacité de la solution. Au départ, aucune commande automatique n'est proposée. La totalité des commandes est donc entrée par l'utilisateur. Plus le fauteuil acquiert d'expérience, moins l'utilisateur n'est sollicité. Au bout de 75 trajets, on arrive à une proportion évoluant autour de 75% de commandes manuelles, ce qui est très satisfaisant. Cependant, l'environnement est statique et l'utilisateur ne commet pas d'erreurs. On est donc dans les conditions optimales.

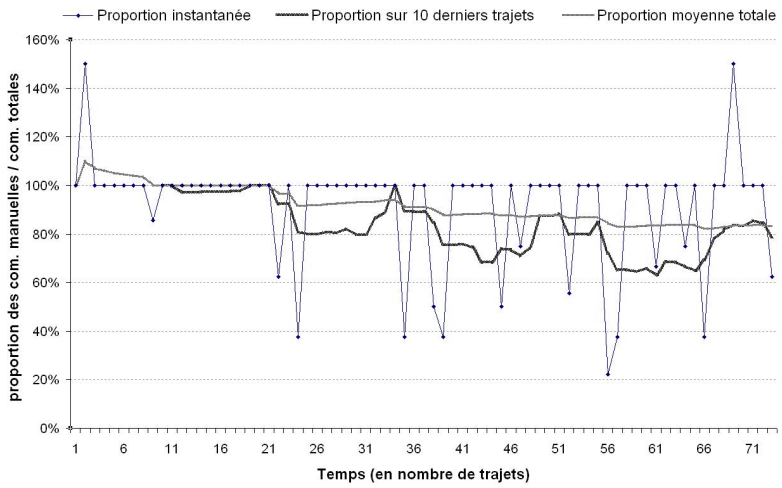


FIGURE 10 – Proportion instantanée et globale de commandes à saisir par l'utilisateur en utilisant notre algorithme par rapport au nombre de commande qu'il aurait eu à saisir en pilotage libre, en fonction du nombre total de trajets parcourus

5 Conclusion et perspectives

Nous avons proposé et testé sur simulateur un algorithme capable d'assister l'utilisateur d'un fauteuil roulant automatique en prédisant les directions à suivre. Les trajets parcourus sont appris sous forme de scénarios dont chaque état correspond à un environnement local du fauteuil. L'algorithme utilisé effectue en continu la maintenance de ses connaissances afin de pouvoir déterminer à tout moment le scénario le plus probable. Les états et scénarios sont appris, ce qui confère une grande autonomie au fauteuil : il s'adapte lui-même à son environnement, à l'utilisateur et aussi aux changements d'habitudes de l'utilisateur.

Les résultats obtenus sont très satisfaisants mais les conditions expérimentales étaient optimales et simulées sur informatique. Nos prochains objectifs sont d'une part d'expérimenter notre solution sur un véritable fauteuil et d'autre part d'étudier le comportement du système en le faisant piloter par un utilisateur novice ne connaissant pas le principe de l'algorithme. Cela permettra d'améliorer ce dernier pour le rendre plus polyvalent et moins sensible aux erreurs de manipulation.

Références

[1] Ali Aich et Sophie Loriette. Saped : System of assistance for people with disabilities. In *ICTAI (1)*, pages 101–108, 2007.

[2] Maria Fox, Malik Ghallab, Guillaume Infantes, et Derek Long. Robot introspection through learned hidden markov models. *Artif. Intell.*, 170(2) :59–113, 2006.

[3] Nir Friedman et Daphne Koller. Being bayesian about network structure. a bayesian approach to structure discovery in bayesian networks. *Machine Learning*, 50(1-2) :95–125, 2003.

[4] Cristina E. Manfredotti. Modeling and inference with relational dynamic bayesian networks. In *Canadian Conference on AI*, pages 287–290, 2009.

[5] Lorraine McGinty et David C. Wilson, editors. *Case-Based Reasoning Research and Development, 8th International Conference on Case-Based Reasoning, ICCBR 2009, Seattle, WA, USA, July 20-23, 2009, Proceedings*, volume 5650 of *Lecture Notes in Computer Science*. Springer, 2009.

[6] Alain Mille. Raisonner à Partir de l'Expérience Tracée, April 2006. "Le storytelling : concepts, outils et applications", sous la direction de Eddie Soulier, Traité IC2, Série Informatique et SI, Hermes Science.

[7] J-M. Nigro et B. Boukabeche. Métaconnaissances et assistance à la conduite en fauteuil roulant. In *HANDICAP*, 2004.

[8] James L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3) :223–252, 1977.

[9] A. Pruski et O. Habert. Obstacle avoidance module for the vahm-2 wheelchair. In *AAATE*, 1999.

Comparaison de la Méthode de Conception Inventive (MCI) basée sur la TRIZ et de l'approche du Raisonnement à Partir de Cas (RàPC)

Denis Cavallucci[1], François Rousselot[1], Jean Renaud[1]
[1] LGECO-INSA Strasbourg

Résumé

Cet article se propose de présenter deux approches utilisant le raisonnement par analogie, et qui font usage d'expériences ou de situations d'expériences passées dans leur mode opératoire. La première, appelée **M**éthode de **C**onception **I**nventive issue de la **TRIZ** a la particularité d'amener le concepteur à raisonner sur des solutions abstraites par rapport au domaine et qui s'adresse plutôt aux problèmes de conception ouverts à l'invention. La deuxième, appelée **RàPC** (Raisonnement à Partir de Cas) a la particularité de prendre en compte des solutions antérieures similaires ou proches pour résoudre des problèmes nouveaux dans tout domaine. Son principe repose sur l'analogie des problèmes. Les auteurs se proposent de présenter ces deux approches pour montrer leurs points de convergence ou non et ainsi observer les apports possibles de l'une à l'autre.

Mots clés : Raisonnement par analogie, Conception Inventive, TRIZ, RàPC

1 Introduction

De nos jours, les entreprises revendiquent de façon quasi unanime être des entreprises innovantes. Une forme intéressante d'innovation réside dans la réutilisation des connaissances ou de l'expérience acquises au cours des activités de conception des produits antérieurs. L'expérience repose de plus en plus sur des connaissances tacites, issues de l'action et de l'apprentissage au cours du temps. Le phénomène d'apprentissage ne consiste plus à diminuer les coûts à partir de courbes d'apprentissage, mais plus à capitaliser un niveau de réflexion conceptuel.

La Méthode de Conception Inventive issue de la TRIZ a la particularité d'amener le concepteur à chercher des solutions abstraites par rapport au domaine. Elle s'adresse plutôt aux problèmes de conception en général. La deuxième méthode appelée Raisonnement à Partir de Cas a la particularité de prendre en compte des solutions antérieures similaires pour résoudre des problèmes nouveaux dans tout domaine. Ces approches ont des similitudes, elles font référence à des solutions antérieures et utilisent le principe de l'analogie des problèmes. Dans ce papier, les auteurs présentent les deux approches et les comparent sur le principe de l'analogie.

2 L'analogie

Selon L. Bérubé, [1] « *L'analogie est une opération intellectuelle par laquelle on étend à l'ensemble d'une classe, ou d'un groupe, les propriétés et caractères observés sur un nombre limité de cas individuels* ». Il est largement reconnu que le raisonnement par analogie est particulièrement difficile à intégrer dans une modélisation, car il ne jouit pas d'un statut scientifique bien établi. Il est plus reconnu dans le cadre des sciences cognitives et de l'intelligence artificielle, comme un mode de raisonnement de la plus grande importance dans la prise de décision, dans les mécanismes d'apprentissage, dans l'heuristique scientifique, dans la vie courante et aussi dans le domaine du droit.

Le raisonnement par analogie est surtout utilisé dans le cadre de **l'apprentissage** des savoirs et savoir-faire au cours des activités de travail. Le raisonnement adopté pour s'approprier ou acquérir un savoir est propre à chacun. Pour tendre vers l'individualisation de l'apprentissage, il convient de proposer des activités

différentes permettant de mener des raisonnements différents pour l'acquisition d'une même compétence. Du point de vue d'un élève, le raisonnement par analogie peut se traduire par « *je reproduis la démarche déjà employée pour un problème présentant des ressemblances, que l'on établit par comparaison entre des éléments différents* ».

L'analogie est un mode de raisonnement souvent pratiqué par l'homme. Une synthèse sur la généralisation de connaissances qui fait le tour des raisonnements par analogie [2] montre que la formalisation et l'informatisation de ce type de raisonnement sont des problèmes difficiles. Généralement, les travaux réalisés en Intelligence Artificielle définissent une famille limitée de représentations sur lesquelles on va pouvoir rechercher des associations que ce soit pour détecter des analogies structurelles [3] ou pour détecter des analogies entre des raisonnements comparables appliqués par exemple à des exercices de géométrie [4]. Les systèmes à raisonnement analogique implémentés sont très spécifiques, ils nécessitent la définition d'un domaine précis et d'un mode de mise en correspondance capable de permettre l'analogie.

Le raisonnement analogique humain n'a pas les limitations de l'informatique, tout se passe comme si l'homme voyait sans difficulté les traits communs à faire correspondre et choisissait parmi les analogies possibles, celle qui donne un résultat pertinent. Enfin, on peut distinguer deux usages du raisonnement analogique : d'une part pour inférer quelque chose, d'autre part pour suggérer des idées ou des concepts nouveaux.

3 Principe et étapes du RàPC

3.1 Le RàPC

Le Raisonnement à Partir de Cas (RàPC) utilise des ***expériences passées validées*** (appelées cas) comme ***solutions*** pour résoudre des ***problèmes nouveaux***. Le RàPC repose sur le principe du raisonnement par analogie [5]. Le principe d'analogie utilisé est le suivant : si deux problèmes sont analogues alors, les solutions associées pour les résoudre sont également analogues.

Le problème « C » est à la solution D ce que le problème « A » est à la solution B. Les relations entre C et D ou A et B sont supposées connues ou non, mais elles existent. Si les problèmes C et A sont analogues ou semblables, une correspondance entre eux est possible. On peut supposer aussi établir une correspondance entre les solutions respectives aux problèmes C et A. Ainsi, la solution du problème A peut convenir à résoudre le problème C. Dans la pratique, on s'aperçoit que l'utilisation de la solution B pour résoudre le problème C ne se fait pas sans un minimum d'adaptation. Ce principe n'est valable que si les problèmes A et C sont supposés de la même classe, pour appliquer les mêmes règles. De même, les solutions sont supposées proches. On représente souvent le raisonnement analogique par le carré analogique, figure 1 .

Après avoir identifié un problème nouveau à résoudre (appelé problème cible pb(cible)), il s'agit de se remémorer ($\alpha_{problème}$) à partir d'un ensemble de problèmes, celui qui sera similaire ou analogue au problème cible, appelé problème source (pb(source)) afin de connaître sa solution préconisée (β_{source}). Puis, il s'agit d'adapter la solution source pour devenir une solution cible ($\alpha_{solution}$) au problème cible. Après validation, le problème cible aura sa propre solution (β_{cible}). Le couple {problème cible, solution cible} pourra être mémorisé dans une base de cas. On appelle cas le couple {Pb, Sol}.

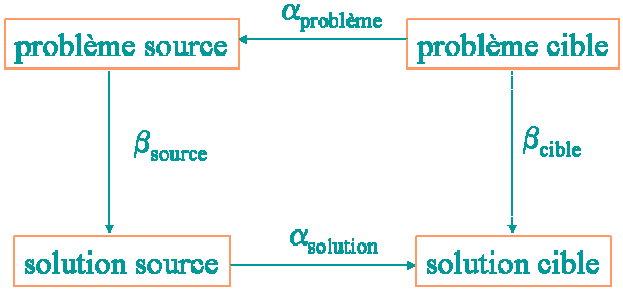


Fig 1.

3.2 . Le processus du RàPC

Le RàPC trouve sa justification dans la psychologie cognitive : la plupart des personnes améliorent leur capacité à résoudre des problèmes par leur expérience. Elles résolvent plus difficilement des problèmes nouveaux que des problèmes rencontrés ou similaires. La base de cas doit se créer par apprentissage afin de limiter le nombre de cas stockés. Le RàPC est un cas particulier du raisonnement par analogie où les cas source et cible ***appartiennent au même domaine***. Le RàPC a de nombreuses applications dans des domaines variés comme la médecine, l'aide au diagnostic, la planification… domaines où les expériences passées représentent une part importante de l'activité.

Les grandes étapes décrivent ce mode de raisonnement :

- ***identification et reconnaissance du problème***. La description du problème va contribuer à la remémoration des expériences similaires. Une description trop sommaire du problème risque de couvrir un espace trop large de problèmes similaires. À l'inverse, une description trop précise risque d'amener à ne trouver aucun problème antérieur. Cette étape suppose de bien connaître le cœur de métier ou le domaine d'études, elle permet d'obtenir le Problème Cible {Pb(cible)} à résoudre
- ***remémoration des expériences passées*** similaires ou analogues associées à leurs solutions. Les fonctions d'appariement seront sollicitées selon le type de remémoration et le type de description du problème. Des méthodes d'indexation et des mesures de similarité seront adaptées aux situations afin d'obtenir la meilleure sélection possible de cas remémorés. La définition d'une mesure de similarité n'est pas triviale.
- ***modification et adaptation de la solution source*** comme solution potentielle capable de résoudre le problème cible. Cette étape est aussi importante que l'étape de remémoration, de plus elle est liée à cette dernière. Plus le cas remémoré est proche ou similaire du cas cible, plus l'effort d'adaptation peut être facilité. Dans certains systèmes, la tâche d'adaptation n'est pas utile. Il existe plusieurs types d'adaptations possibles : par ***substitution***, cette forme se limite à modifier la valeur d'un attribut élémentaire ; par ***transformation***, on transforme la structure ainsi que les valeurs des attributs du cas ; par ***dérivation***, la solution du cas retenu est adaptée en réexécutant des parties originelles de la solution.
- ***test et évaluation de la nouvelle solution.*** Cette étape va valider la pertinence de la solution remémorée et adaptée au nouveau problème. Une attention particulière sera opérée afin de connaître le niveau de pertinence de cette nouvelle solution.
- ***apprentissage et mémorisation du nouveau cas {pb(cible), sol(cible)}***.Cette étape va enrichir la base de cas et permettre de résoudre de nouvelles situations ou problèmes. Le nouveau cas devra apporter une compétence ou une valeur ajoutée aux cas déjà remémorés. Les cas devront également respecter la même granularité de connaissances. Enfin, une organisation adéquate de la base de cas (indexation…) sera soignée dans le but de faciliter l'étape de remémoration.

4 La Méthode de Conception Inventive

4.1 Introduction à TRIZ

La théorie TRIZ résulte d'un travail collaboratif d'un ensemble de chercheurs et ingénieurs de l'ex-URSS coordonnés dans leurs activités par G. Altshuller [6]. Après 60 années de développement, elle propose aujourd'hui un ensemble de postulats, axiomes, lois, techniques et processus utiles pour aborder une situation de conception ouverte à l'invention. La TRIZ s'appuie sur une analyse issue de la dialectique pour créer son modèle principal qui repose sur la détection des contradictions [7]. Selon la dialectique, toute organisation, société ou situation évolue en résolvant ses contradictions, chaque nouvel état amène bien sûr d'autres contradictions qui à leur tour devront être résolues. Appliquée aux systèmes techniques, la dialectique a donné naissance dans la TRIZ à une théorie de l'évolution de ces systèmes encodée dans des lois dites "lois d'évolution".

Le principal obstacle à l'invention est l'inertie psychologique qui nous fait rechercher des solutions uniquement dans les domaines que l'on maîtrise, où nous sommes experts. La TRIZ donne les moyens de vaincre cet obstacle, elle pousse à construire des modèles abstraits par rapport à un domaine particulier de manière à faciliter la recherche de solutions éventuelles dans d'autres domaines. En effet, l'étude de milliers de brevets a permis d'identifier un certain nombre de couples {problème abstrait, solution abstraite} qui correspondent à des couples {problèmes, solutions} de domaines très différents.

Le principe de la TRIZ est souvent présenté comme un processus qui part d'une situation concrète pour construire un modèle abstrait par rapport au domaine (processus montant), puis qui associe à ce modèle abstrait une solution abstraite. Finalement, il convient d'évaluer cette solution abstraite dans le domaine spécifique de l'application en cours (processus descendant) pour proposer une solution concrète.

La TRIZ fournit des directives au concepteur pour la construction du modèle de la situation à traiter, puis lui propose des bases de connaissances qui le conduisent vers un (ou des) " concept (s) de solution ". Un concept de solution est une solution abstraite qu'il conviendra de valider dans le domaine.

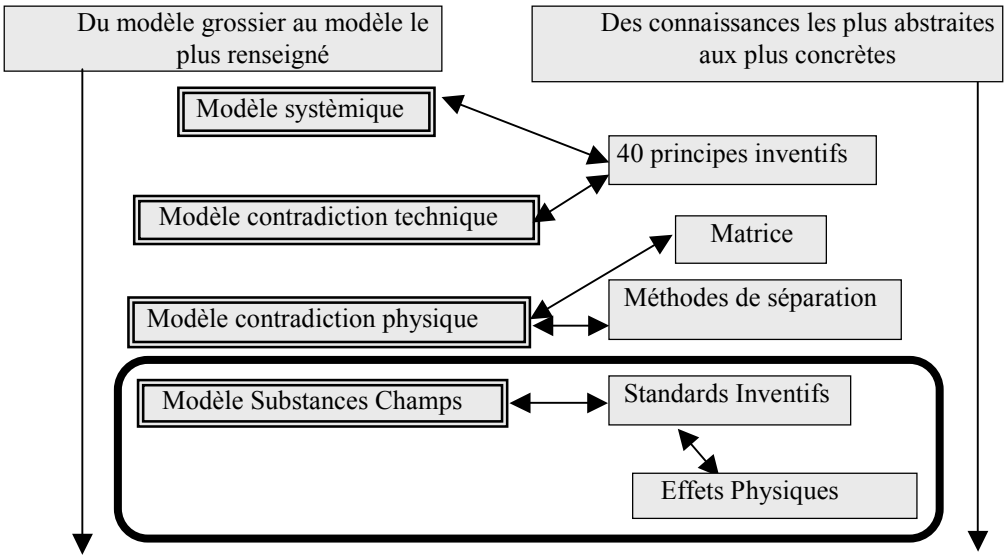


Fig 2 Les différents modèles de la conception inventive

Plus précisément, la TRIZ propose plusieurs modèles représentés Fig 2 (rectangles aux bordures doubles), situés à des niveaux d'abstraction différents. Les flèches descendantes indiquent que le niveau d'abstraction diminue et se rapproche du niveau du domaine.

Dans la suite de cet article, nous nous focaliserons sur le modèle de plus bas niveau (Substance-Champs) et la résolution du problème à ce niveau d'abstraction (cadre en gras sur la figure).

Le premier modèle appelé **modèle systémique** a pour rôle d'isoler la fonction principale utile du système et les éléments qui la composent (contrôle, énergie, outil, produit, etc.).

Les deux modèles suivants concernent la contradiction. La TRIZ distingue deux types de contradictions et propose pour résoudre celles-ci, plusieurs bases de connaissances

a) La **Contradiction technique** : l'évolution de la valeur d'un paramètre A dans le sens d'une amélioration, conduit à la dégradation de la valeur d'un autre paramètre utile B (et réciproquement). Les contradictions techniques sont résolues en utilisant la matrice de résolution et les 40 principes inventifs.

Le deuxième modèle est donc celui de la contradiction technique

b) La **Contradiction physique** : contradiction où un paramètre d'un élément du système doit présenter deux valeurs contradictoires à la fois[1] comme fort et faible, dur et mou, lisse et rugueux (en général, ces

[1] On appelle paramètre d'action un paramètre sur lequel, on peut agir, et paramètre d'évaluation un paramètre qu'on peut simplement évaluer.
La contradiction se caractérise par un ensemble de 3 paramètres (2 paramètres d'évaluation et 1 paramètre d'action) tels que si PA_1 est à Va alors PE1 évolue dans le sens recherché et PE_2 dans un sens non recherché. Mais que si PA_1 est à Vā alors PE_1 évolue dans le sens non recherché et PE_2 dans le sens recherché.

contradictions sont résolues par des méthodes de séparation).

Le troisième modèle (de la contradiction physique) fait ressortir la **contradiction** principale à la base du problème spécifique à l'application.

La détermination de la contradiction principale est la clé de la compréhension du problème.

Le quatrième modèle **le Modèle Substances-Champs** (MSC dans la suite, parfois encore appelé "vépole") oriente l'utilisateur vers l'utilisation de deux sources de connaissances : les "**standards**", et venant en complément : les " **effets physiques** ". Le MSC est instancié par les éléments : Substances-Champs de la situation concrète, il est donc proche des réalités physiques et permet de trouver des concepts de solutions proches de solutions concrètes.

Le MSC nécessite une connaissance fine de la méthode lors de sa construction. Ce modèle amène par sa résolution à produire des **concepts de solution** plus faciles à évaluer que les concepts de solutions abstraits engendrés par les principes par exemple.

En nous focalisant ici sur le modèle MSC, nous nous démarquons de plusieurs recherches effectuées sur le thème de la comparaison de TRIZ et du RàPC. En effet, celles-ci sont axées sur l'utilisation de la matrice et des principes [8,9] et considèrent généralement que l'application d'un ou plusieurs principes est une solution. Or, le niveau d'abstraction des principes est relativement éloigné d'une solution concrète du monde réel.

Construction du modèle

En résumé, le processus de construction du modèle est primordial. Il faut en effet que le modèle prenne en compte tous les éléments de la situation initiale pour que la résolution conduise à une bonne solution. L'enjeu est de comprendre cette situation dans toute sa multidisciplinarité, il faut donc collecter tous les éléments pouvant y contribuer en questionnant les experts, en analysant des textes de référence, des textes de brevets, etc. Le premier postulat de la TRIZ stipule que les systèmes techniques sont assujettis à des lois objectives, les **lois d'évolutions** qui, formulées de façon générique [10], donnent des indications aux concepteurs sur les directions à prendre en priorité pour accompagner l'objet dans son évolution. Le modèle obtenu à l'issue de la phase d'analyse est souvent très éloigné de la formulation initiale du problème. Il met en évidence les contradictions principales (3 contradictions maximum en TRIZ) à la source du problème qui bloque son évolution.

4.2. Présentation de la Méthode de Conception Inventive

La Méthode de Conception Inventive (MCI) étend le champ de la TRIZ à des problèmes complexes[2] [11]. Chaque contradiction exprime un problème qui provient de l'incompatibilité entre des valeurs de paramètres dans une certaine zone appelée **zone opératoire** durant un certain temps : l'**intervalle opératoire**.

Cette zone détermine les **substances** et les **interactions** (appelées champ en TRIZ) qui sont à l'origine de la contradiction : cette focalisation permet d'identifier les substances en jeu, les interactions entre ces substances et notamment les interactions non désirées. Une fois la collecte des substances et des interactions en jeu effectuée, le Modèle Substances-Champs qui dérive de la contradiction peut être construit. Cette phase de modélisation a pour objectif de ramener une situation particulière en termes de contradictions à un modèle générique exprimé par des faits physiques qui utilise seulement trois types d'entités :

- **Les substances** : Solides, Liquides, Gaz ou Plasmas ;
- **Les champs** : Mécaniques, Chimiques, Thermiques, Electriques ou Magnétiques ;
- **Les interactions** : utiles, excessives, insuffisantes ou nuisibles.

Les standards et leur utilisation

Comme leur nom l'indique, les standards encodent la connaissance capitalisée en TRIZ autour des problèmes classiques abstraits résolus dans les brevets dits "***problèmes standards***" ou "***standards***". Ils sont

[2] TRIZ considère que la ou les deux ou trois contradictions principales sont identifiées et se focalise sur la résolution de celles-ci. Les problèmes complexes comportent des centaines de contradictions.

utilisés pour la résolution des problèmes posés en termes de MSC, ils ont la forme de règles spécifiques. Ces règles "***si-alors***" sont différentes de celles qu'on trouve en Intelligence Artificielle. En effet, leurs prémisses portent sur la situation de problème et permettent de préciser par des questions posées au concepteur, quels changements peuvent être opérés à son avis sur le modèle qui décrit la situation initiale du problème.Quand la prémisse d'une règle est vraie, la règle indique dans son membre droit les actions à effectuer pour modifier le modèle. La modification du modèle engendre alors un modèle de problème générique dont qui a une solution connue.

La base de connaissances des standards contient une liste de couples {modèle de problème, modèle de solutions} ainsi que la description des transformations décrivant comment passer du modèle d'une situation quelconque à un modèle de problème générique.

Les problèmes standards décrivent une typologie réduite de modèles incomplets ou ne "fonctionnant" pas suffisamment bien. À partir d'un modèle qui montre un dysfonctionnement, les transformations vont naturellement viser à supprimer ce défaut. Si par exemple, le dysfonctionnement porte sur une substance qui manque, un champ qui est insuffisant ou si encore un autre champ qui a un effet nuisible intervient..., les transformations correspondantes vont avoir pour effet respectivement de rajouter la substance manquante, de renforcer le champ, de supprimer le champ nuisible... Pour chaque modèle incomplet, les standards énumèrent plusieurs actions réparatrices dans un ordre déterminé privilégiant les actions les moins coûteuses.

Dans une application classique (manuelle) de la méthode, toutes les transformations sont effectuées de façon systématique et ne tiennent pas compte du contexte ce qui rend l'application de la méthode fastidieuse. Dans [12], nous montrons comment nous avons "décompilé" la connaissance des standards pour en faire des bases de connaissances plus lisibles et extensibles. Dans notre système de représentation des connaissances, nous avons représenté toute transformation quelle qu'elle soit, par un couple {état initial du MSC, état final du MSC} comme il est classique de la faire en Intelligence Artificielle.

Nous présentons plus bas Fig3. un exemple d'utilisation du MSC pour la résolution d'un problème particulier.

Un exemple

Soit un système technique où de l'eau sous forte pression passe dans un tuyau, cette forte pression engendre un effet nuisible au voisinage d'un coude qui se manifeste par des coups de boutoir qui menacent de rompre le tuyau.

Le schéma qui représente la situation initiale : au centre et à gauche (fig.2) résume les substances et les champs en jeu. Les flèches pleines représentent des interactions utiles et les flèches en pointillés les interactions nuisibles. Il s'agit ici d'une représentation graphique destinée à faciliter la compréhension. Dans notre système informatique, ces modèles sont représentés dans un formalisme centré objet, les raisonnements principaux sont d'ailleurs effectués par un système de logique de description.

Ce schéma décrit le passage d'un modèle de situation initiale où apparaît un champ nuisible, au milieu à gauche, vers le modèle de problème générique correspondant (standard). Le standard correspondant a demandé à l'utilisateur s'il est envisageable de modifier la substance S1 (c'est possible ici). La solution possible consiste en fait à trouver une modification de S1 qui supprime le champ nuisible. Le passage du modèle de la situation initiale au modèle abstrait est symbolisé par les grosses flèches verticales qui indiquent le sens de l'abstraction.

Une fois le couple {problème abstrait, solution abstraite} trouvé, le système parcourt alors la base des effets physiques pour voir si une loi, un effet physique permet de réaliser ce qu'on cherche. Ici, il s'agit l'effet dit de "cavitation acoustique"[3] qui permet de supprimer un champ mécanique nuisible, c'est le " **concept de solution** " trouvé.

Les flèches orientées à midi moins dix indiquent que l'appariement de la transformation correspondant à l'effet physique cavitation acoustique est envisageable, car il est possible d'établir une correspondance entre

[3] La cavitation acoustique consiste à créer un champ acoustique au voisinage du tuyau, celui-ci aura pour effet de créer des sortes de bulles dans le liquide qui amortiront les chocs d'impacts.

les substances et les interactions des couples générique et effet physique.

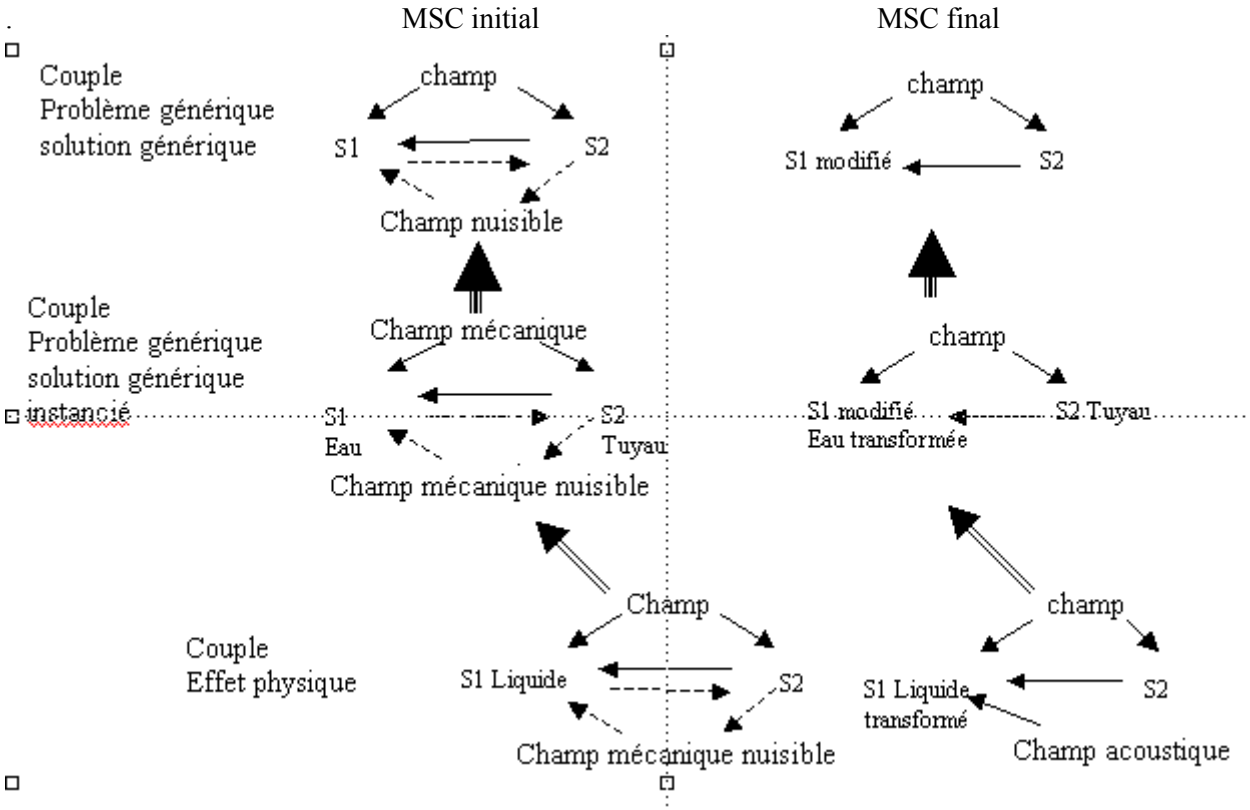


Fig 3. Un exemple de raisonnement basé sur le MSC

On n'obtient pas une solution du domaine, mais une solution générique instanciée : il s'agit d'un concept abstrait de solution, qu'il faut valider (c.-à-d. voir dans le domaine si l'idée est réalisable).

Le déroulement du processus n'est pas terminé, les différents questionnements induits par les standards permettent d'envisager un certain nombre de problèmes génériques et pour certains de générer d'autres concepts de solutions.

L'analyse MSC est destinée à fournir plusieurs concepts de solutions qui seront ensuite évalués techniquement et économiquement par les experts du domaine en tenant compte du contexte. L'effort d'interprétation est fonction de la clarté de l'expression générique du standard complété et du savoir de l'expert, il nécessite d'être créatif. Aussi un concepteur non créatif risque de ne pas trouver d'interprétation possible au standard.

5 Comparaison sommaire MCI RàPC

Le modèle de raisonnement de la TRIZ, considéré dans notre article est « analogue » à celui du RàPC, car il part lui aussi d'un problème cible pour tenter de faire émerger une solution en terme de résultat attendu.

5.1 Préparation des systèmes

Avant le déroulement d'un problème particulier, il convient de parler de la préparation du système.

Le RàPC : l'investissement est important pour chaque domaine envisagé, il faut répertorier les caractéristiques importantes permettant d'effectuer le raisonnement et également entrer des cas initiaux.

La MCI: elle utilise les bases de connaissances répertoriées dans le cadre de la TRIZ qui sont génériques et ne dépendent d'aucun domaine technique particulier. Les bases de connaissances utilisées pour la résolution notamment les effets physiques doivent être mises à jour en fonction des découvertes de la

physique.

En ce qui concerne le déroulement des deux processus, il y a des phases où les deux approches convergent et d'autres où elles diffèrent. Nous allons maintenant passer en revue les deux approches en partant du carré analogique du RàPC afin d'y associer un schéma analogique propre à la démarche MCI.

Fig 4. Les phases des deux méthodes de raisonnement (RàPC et la TRIZ)

5.2 Les phases du raisonnement

5.2.1 Première phase : Caractérisation du problème cible

Le RàPC : Il définit un problème cible comme étant le problème objet de la résolution. Ce dernier est souvent construit à partir d'une description problème pendant une phase appelée « élaboration ».

La MCI: Dans une étape analogue, on travaille à la reformulation à partir de la situation et des connaissances initiales. Mais la différence majeure ici est que la reformulation aboutit à un problème cible qui traduit une hypothèse d'évolution (mener une action de conception sur un objet de départ pour accompagner son passage d'une génération vers une autre).

Mesure de similarité

Le RàPC : Elle tient compte de l'influence d'une variation de valeur du descripteur problème sur une variation de valeur d'un descripteur solution.

La MCI: les modèles étant abstraits et génériques, une mesure de similarité est inutile, à deux problèmes réels semblables correspond le même problème générique.

5.2.2. Deuxième phase : Problème source

Le RàPC : Il définit un problème source comme étant le problème qui, recherché dans une base d'exemples, partagera des caractéristiques voisines à celle du problème cible.

La MCI: Dans cette étape, on construit un MSC à partir du modèle de la contradiction. Ainsi, ce qui au départ était un problème cible devient un problème source défini par un Modèle Substances-Champs.

Relations de dépendance

Le RàPC : Elles permettent de mettre en évidence les descripteurs solution qui doivent être adaptés car dépendant de descripteurs du problème-source différents des descripteurs du problème-cible.

La MCI: Une analogie peut être faite par rapport aux relations de dépendance entre un MSC exprimant la contradiction et l'ensemble réduit de standards qui lui correspondent. Cette relation se fait entre autres au travers d'un algorithme élaguant les standards au fur et à mesure du questionnement qu'il renferme pour ne faire émerger que ceux qui pourraient correspondre au problème de départ. Les standards résument les relations entre situations problèmes génériques et situations résolues génériques. Ces relations permettent à l'utilisateur de retrouver les façons les plus inventives employées par les inventeurs pour résoudre leur problème lorsqu'ils étaient confrontés à des situations de problèmes analogues au problème source.

5.2.3 Troisième phase : Solution source

Le RàPC : Une solution source en RàPC est une solution appartenant à une base de cas déjà existante.

La MCI: Cette étape est directement reliée à l'expression d'un standard qui permet de construire une solution source. Il y a ici une différence notoire quant à la précision des solutions sources. Si en RàPC elles sont précisément définies, elles sont volontairement génériques dans la MCI (ou la TRIZ) pour être représentatives de situations « résolues de façon inventive » exprimées de façon abstraite, mais dont la forme syntaxique permet la précision a posteriori.

Inférence analogique

Le RàPC : Il s'agit ici de calculer le degré d'appartenance de la solution source au problème cible à partir de sa similarité.

La MCI: Dans cette étape, la MCI ne calcule pas de solution, mais fait appel aux mécanismes cognitifs d'association que tout être humain possède dès lors que l'expression générique d'une solution lui rappelle une situation analogue issue d'une expérience vécue par ailleurs. Elle préconise pour cela de compléter le modèle d'un standard isolé en remplaçant dans un premier temps les expressions génériques S3, C1 par les substances et les champs reconnus comme produisant les effets recherchés. L'analogie peut être faite avec une équation dont on rechercherait les inconnues (X, Y) et dont les attributs cibles auraient été consignés. Ces attributs trouvent des populations d'équivalents dans des bases d'effets physiques qui viennent compléter le modèle du standard inventif par des expressions plus proches d'une réalité physique favorisant l'émergence d'un Concept de Solution.

5.2.4 Quatrième phase : Solution cible

Le RàPC : Elle caractérise le résultat attendu du cas, l'identification du meilleur candidat pour devenir la solution du cas traité.

La MCI : Alors que le RàPC vise une solution au plus proche de la solution source, la MCI offre un ensemble réduit de concepts de solutions possibles. Ces derniers sont autant d'orientations, de populations de solutions pouvant potentiellement résoudre le problème. En règle générale, les quelques standards complétés par les effets physiques sont déclencheurs de concepts dans l'esprit des concepteurs qui en font usage et capitalisent ces concepts pour ensuite décider des pistes de développement à effectuer en R&D.

6 Conclusion

Résumons les points de divergences des deux approches :

Point 1 **: Autoriser ou non la divergence**

Le RàPC ne s'éloigne jamais des attributs qui caractérisent une solution d'un problème. Ceux-ci déterminent les inférences qui serviront à rapprocher les deux mondes par des calculs quantitatifs à l'intérieur du domaine.

La MCI, en passant par des modèles parfois qualitatifs et en donnant une part importante à l'expérience de l'utilisateur de la méthode, favorise l'éloignement par rapport au domaine d'origine.

Point 2 **: Les modèles de référence ne sont pas au même niveau systémique**

La capitalisation des cas dans la MCI ne nécessite à aucun moment d'être réeffectuée. Elle fixe un état générique de représentation des connaissances fondamentales détaché de toute discipline particulière. Ceci permet d'aller vers l'invention c.-à-d. le(s) concept(s) de solution lors des échanges disciplinaires, alors que la représentation paramétrée d'un même domaine qui caractérise le RàPC favorise l'émergence d'une solution du domaine lorsqu'elle existe.

Point 3 : Ce qui caractérise le problème cible

La plus grande différence entre les approches est sûrement l'orientation de départ donnée à l'étude. Si le RàPC caractérise un problème cible sur des bases diverses (volonté cliente, identification d'un dysfonctionnement, recherche de cause) la MCI considère que ce problème cible se doit d'être une direction d'évolution induite par les lois d'évolution de la TRIZ suite à une étude de la maturité de l'objet étudié. Ici, la règle est souvent de suivre plutôt une orientation d'évolution particulière que de se plier à la volonté stratégique des acteurs de l'étude.

En conclusion, vu les différences de point de vue entre les deux approches, il semble assez difficile de les faire collaborer. Notamment, s'il est concevable de rajouter une capitalisation des cas nouveaux résolus en MCI, pour mémorisation et rappel, il faut voir qu'ils ne seront pas utilisables au même titre qu'en RàPC. Cela irait à l'encontre de l'esprit de la conception inventive. Ces cas ne seront utilisables que dans la phase ultime d'évaluation des concepts de solutions trouvés, lorsqu'on étudie la faisabilité des concepts de solutions dans le domaine.

7 Références

[1] L. Bérubé, (tiré de Terminologie de neuropsychologie et de neurologie du comportement. Recherche et réd. Louise Bérubé., 176 p., 1991, reproduit avec la permission des Éditions de la Chenelière Inc., p.17)

[2] E. Cauzinille-Marmèche, J. Mathieu.Généralisation des connaissances et résolution de problèmes. In:*L'année psychologique*. 1994 vol. 94, n°3. 461-484, 1994

[3] D.Gentner Structure-mapping: A theoretical framework for analogy. *Cognitive Science,* 7, 155-170. 1983

[4].E. Chouraqui, C. Inguilterra Résolution par analogie de problèmes géométriques dans une perspective tutorielle Proc. of the Second International Conference on Intelligent Tutoring Systems *Lecture Notes In Computer Science*; Vol. 608: 156 - 163 1992 ISBN:3-540-55606-0

[5] M. Py. Un modèle conceptuel de *raisonnement* par *analogie*. Revue d'Intelligence Artificielle, 8:63–. 99, 1994.

[6] G. Altshuller, B. Zlotin, A. Zusman, A. & V. Philatov., Tools of classical TRIZ, Ideation Int. Inc, ISBN 1-928747-02-7, Mars 1999.

[7] C. Zanni-Merk, P. Bouché, Dialectics-based Knowledge Acquisition – A case study, ISBN : 978-3-642-04594-3, KES 2009 13th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, 28-30 Septembre 2009

[8] G. Cortes Robles, E. Cordova Lopez, S. Negny, J-M. Le Lann G. Lacoste, L'innovation industrielle par la méthode Triz : amélioration d'un chaîne de production, 5e Conférence Française de Modélisation et Simulation, MOSIM' 04, Nantes. 2004

[9] I. Estevez, S. Dubois, N. Gartiser, J. Renaud, E. Caillaud Le raisonnement à partir de cas est-il utilisable pour l'aide à la conception inventive 14° Atelier de Raisonnement à Partir de Cas, Besançon (France) 30-31mars, 2006.

[10] Y. Salamatov Triz: the Right Solution at the Right Time: A Guide to Innovative Problem Solving ISBN 9080468010 Ed.Insytec BV . New edition 1999.

[11] F. Rousselot, C. Zanni-Merk, D. Cavallucci, **'**Une ontologie pour l'acquisition et exploitation des connaissances en conception inventive**'**, in Revue des Nouvelles Technologies de l'Information, Vol.E n°19,

717-738, ISBN : 978-2-85428-922-0, Cépaduès-Éditions, 12 Février 2010

[12] A. Bultey, C. Zanni-Merk, F. Rousselot, F.de Beuvron A hybrid system combining description logics and rules for inventive design KES 2009

Un algorithme d'adaptation avec des cas exprimés dans la logique de descriptions $\mathcal{ALC}$

Julien Cojan, Jean Lieber
UHP-Nancy 1 – LORIA (UMR 7503 CNRS-INPL-INRIA-Nancy 2-UHP)
BP 239, 54506 Vandœuvre-lès-Nancy, France
{Julien.Cojan, Jean.Lieber}@loria.fr

Résumé

Cet article décrit un algorithme d'adaptation pour un système de raisonnement à partir de cas où les cas et les connaissances du domaine sont exprimés dans la logique de descriptions expressive $\mathcal{ALC}$. Le principe consiste à supposer que le cas source à adapter résout le cas cible, ce qui entraîne des contradictions avec le contexte du cas cible et avec les connaissances du domaine. L'adaptation consiste alors à résoudre ces contradictions. L'algorithme est fondé sur une extension de la méthode classique des tableaux employée pour le calcul d'inférences déductives en $\mathcal{ALC}$.

Mots clés : adaptation, logique de descriptions, $\mathcal{ALC}$, algorithme des tableaux

1 Introduction

L'adaptation est une étape de certains systèmes de raisonnement à partir de cas (RÀPC) qui consiste à modifier un cas source pour qu'il réponde à une nouvelle situation, le cas cible. Une approche de l'adaptation consiste à utiliser un opérateur de révision des connaissances[1], i.e., un opérateur qui modifie de façon minimale un ensemble de connaissances afin d'être cohérent avec d'autres connaissances [1]. L'idée est de considérer la connaissance « Le cas source résout le problème cible » et de réviser cette connaissance par les contraintes données par le cas cible et les connaissances du domaine. Cela a été étudié pour des cas représentés en logique propositionnelle dans [12]. Puis, cela a été étudié dans un formalisme plus expressif, incluant des contraintes numériques [3] et, après cela, étendu à la problématique de combinaison des cas dans ce formalisme [4].

Dans cet article, cette approche de l'adaptation est étudiée pour les cas représentés dans la logique de descriptions (LD) $\mathcal{ALC}$ (qui est la plus simple des « LD expressives »). Le choix des LD comme formalismes pour le RÀPC peut être motivé de plusieurs façons. D'abord, elles étendent les formalismes attributs-valeurs, qui sont souvent utilisés pour le RÀPC (voir par exemple [11]) et ils sont similaires au formalisme des *memory organisation packets* (MOPs) utilisé dans certaines des premières applications du RÀPC [13]. Plus généralement, elles sont conçues comme des compromis entre l'expressivité et la « complexité pratique[2] ». Ensuite, elles ont une sémantique bien définie et elles sont étudiées systématiquement depuis plusieurs décennies maintenant. Enfin, plusieurs implantations efficaces sont disponibles gratuitement, offrant des services qui peuvent être utilisés par les systèmes de RÀPC, en particulier pour la remémoration et l'organisation de la base de cas.

La suite de l'article est organisée comme suit. La section 2 présente la LD $\mathcal{ALC}$ et, en particulier, l'algorithme des tableaux, qui est à la base des inférences déductives pour la plupart des implantations actuelles. Un exemple est présenté dans cette section pour illustrer des notions relativement complexes pour un lecteur qui ne serait pas familier des LD. Cet algorithme des tableaux est étendu pour effectuer un processus d'adaptation, ce qui fait l'objet de la section 3. La section 4 discute notre contribution et présente d'autres travaux utilisant

[1] Le terme anglais est *belief revision*, qui se traduit littéralement par « révision des croyances », que nous préférons traduire par « révision des connaissances », parce que, d'une part, le terme « croyance » a des connotations religieuses hors de propos et, d'autre part, que la distinction en débat *belief-knowledge* sort du cadre de nos travaux.

[2] La complexité dans le pire cas des LD expressives est très élevée, puisque la plupart des inférences sont PSPACE-difficiles. Cependant, en pratique, cette complexité reste raisonnable.

les LD pour le RÀPC, ainsi que d'autres travaux proches. La section 5 conclut l'article et présente quelques perspectives.

2 La logique de descriptions $\mathcal{ALC}$

Les logiques de descriptions [2] forment une famille de logiques classiques équivalentes à des fragments décidables de la logique du premier ordre (L1O). Elles prennent une importance grandissante en représentation des connaissances. $\mathcal{ALC}$ est la plus simple des *LD expressives*, c'est à dire des LD qui étendent la logique propositionnelle. L'exemple culinaire présenté dans cette section est inspiré par le *Computer Cooking Contest*.

2.1 Syntaxe

Les élément du langage de représentation de $\mathcal{ALC}$ sont les concepts, les rôles, les instances et les formules.

Intuitivement, un *concept* représente un sous-ensemble du domaine d'interprétation. Un concept est soit un *concept atomique* (c.-à-d. un nom de concept), ou une expression conceptuelle de l'une des forme suivantes : $\top$, $\bot$, $\neg\mathtt{C}$, $\mathtt{C} \sqcap \mathtt{D}$, $\mathtt{C} \sqcup \mathtt{D}$, $\exists\mathtt{r}.\mathtt{C}$, et $\forall\mathtt{r}.\mathtt{C}$, où C et D sont des concepts (atomiques ou non) et r est un rôle. À un concept peut être associé une formule du premier ordre avec une variable libre x. Par exemple, au concept

$$\mathtt{Tarte} \sqcap \exists\mathtt{ing}.\mathtt{Pomme} \sqcap \exists\mathtt{ing}.\mathtt{Pâte} \sqcap \forall\mathtt{ing}.\neg\mathtt{Cannelle} \tag{1}$$

peut être associé la formule du premier ordre

$$\mathtt{Tarte}(x) \wedge (\exists y, \mathtt{ing}(x,y) \wedge \mathtt{Pomme}(y)) \wedge (\exists y, \mathtt{ing}(x,y) \wedge \mathtt{Pâte}(y)) \wedge (\forall y, \mathtt{ing}(x,y) \Rightarrow \neg\mathtt{Cannelle}(y))$$

Intuitivement, un *rôle* représente une relation binaire. Dans $\mathcal{ALC}$ les rôles sont atomiques, c.-à-d. des noms de rôles. Leur pendant en L1O sont les prédicats binaires. Le rôle apparaissant dans (1) est ing.

Intuitivement, une *instance* représente un élément du domaine d'interprétation. Dans $\mathcal{ALC}$ les instance sont atomiques, c.-à-d. des noms d'instances. Leur pendant en L1O sont les constantes.

Il y a quatre types de formules dans $\mathcal{ALC}$ (suivies par leurs significations) : (1) $\mathtt{C} \sqsubseteq \mathtt{D}$ (C est plus spécifique que D), (2) $\mathtt{C} \equiv \mathtt{D}$ (C et D sont des concepts équivalents), (3) $\mathtt{C}(\mathtt{a})$ (a est une instance de C), et (4) $\mathtt{r}(\mathtt{a},\mathtt{b})$ (r relie a à b), où C et D sont des concepts, a et b sont des instances, et r est un rôle. Les formules de types (1) et (2) sont appelées des *formules terminologiques*. Les formules de types (3) et (4) sont appelées des *formules assertionnelles*, ou simplement des *assertions*.

Une base de connaissances BC en $\mathcal{ALC}$ est un ensemble de formules $\mathcal{ALC}$. La partie terminologique (ou *TBox* pour *terminological box*) de BC est l'ensemble de ses formules terminologiques. La partie assertionnelle (ou *ABox* pour *assertional box*) de BC est l'ensemble de ses formules assertionnelles.

Par exemple, la TBox suivante représente les connaissances du domaine (CD) de notre exemple (avec en commentaire les significations) :

$$\begin{aligned}\mathtt{CD} = \{&\mathtt{Pomme} \sqsubseteq \mathtt{Piridion}, && \mathtt{Poire} \sqsubseteq \mathtt{Piridion}, && \textit{Les pommes et les poires sont des piridions.}\\ &\mathtt{Piridion} \sqsubseteq \mathtt{Pomme} \sqcup \mathtt{Poire}\} && && \textit{Un piridion est soit une pomme soit une poire.}\end{aligned} \tag{2}$$

Notez que la dernière formule est une simplification : en réalité, il y a d'autres piridions que des pommes et des poires. Cette simplification rendra l'exemple plus facile à appréhender mais, comme cela sera expliqué dans la note 7, on pourrait s'en passer.

Dans notre exemple, les seuls cas considérés sont le cas source et le cas cible. Ils sont représentés dans l'ABox suivante :

$$\{\mathtt{Source}(\sigma), \mathtt{Cible}(\gamma)\} \tag{3}$$

$$\text{avec} \quad \mathtt{Source} = \mathtt{Tarte} \sqcap \exists\mathtt{ing}.\mathtt{Pâte} \sqcap \exists\mathtt{ing}.\mathtt{Pomme} \qquad \text{et} \qquad \mathtt{Cible} = \mathtt{Tarte} \sqcap \forall\mathtt{ing}.\neg\mathtt{Pomme} \tag{4}$$

Le cas source est alors représenté par l'instance σ, qui est une tarte avec les types d'ingrédients pâte et pomme. Le cas cible est représenté par l'instance γ déclarant qu'une tarte sans pomme est demandée.

2.2 Sémantique

Une interprétation est un couple $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$ où $\Delta_\mathcal{I}$ est un ensemble non vide (le *domaine d'interprétation*) et où $\cdot^\mathcal{I}$ associe à un concept $\mathtt{C}$ un sous-ensemble $\mathtt{C}^\mathcal{I}$ de $\Delta_\mathcal{I}$, à un rôle $\mathtt{r}$ une relation binaire $\mathtt{r}^\mathcal{I}$ sur $\Delta_\mathcal{I}$ (pour $x, y \in \Delta_\mathcal{I}$, x est lié à y par $\mathtt{r}^\mathcal{I}$ est noté $(x, y) \in \mathtt{r}^\mathcal{I}$) et, à une instance $\mathtt{a}$ un élément $\mathtt{a}^\mathcal{I}$ de $\Delta_\mathcal{I}$.

Étant donné une interprétation $\mathcal{I}$, les différents types d'expressions conceptuelles sont interprétées par :

$$\top^\mathcal{I} = \Delta_\mathcal{I} \qquad \bot^\mathcal{I} = \emptyset \qquad (\neg \mathtt{C})^\mathcal{I} = \Delta_\mathcal{I} \setminus \mathtt{C}^\mathcal{I} \qquad (\mathtt{C} \sqcap \mathtt{D})^\mathcal{I} = \mathtt{C}^\mathcal{I} \cap \mathtt{D}^\mathcal{I} \qquad (\mathtt{C} \sqcup \mathtt{D})^\mathcal{I} = \mathtt{C}^\mathcal{I} \cup \mathtt{D}^\mathcal{I}$$
$$(\exists \mathtt{r}.\mathtt{C})^\mathcal{I} = \{x \in \Delta_\mathcal{I} \mid \exists y, (x, y) \in \mathtt{r}^\mathcal{I} \text{ et } y \in \mathtt{C}^\mathcal{I}\} \qquad (\forall \mathtt{r}.\mathtt{C})^\mathcal{I} = \{x \in \Delta_\mathcal{I} \mid \forall y, \text{ si } (x, y) \in \mathtt{r}^\mathcal{I} \text{ alors } y \in \mathtt{C}^\mathcal{I}\}$$

Par exemple, si $\mathtt{Tarte}^\mathcal{I}$, $\mathtt{Pomme}^\mathcal{I}$, $\mathtt{Pâte}^\mathcal{I}$, et $\mathtt{Cannelle}^\mathcal{I}$ sont les ensembles de tartes, de pommes, de pâtes et de cannelles, et si $\mathtt{ing}^\mathcal{I}$ est la relation « a pour ingrédient », alors le concept de l'équation (1) représente l'ensemble des tartes avec des pommes et de la pâte, mais sans cannelle.

Étant donné une formule f et une interprétation $\mathcal{I}$, « $\mathcal{I}$ satisfait f » est dénoté par $\mathcal{I} \models f$. Un modèle de f est une interprétation $\mathcal{I}$ satisfaisant f. La sémantique des quatre types de formules est la suivante :

$$\begin{array}{lll} \mathcal{I} \models \mathtt{C} \sqsubseteq \mathtt{D} & \text{si } \mathtt{C}^\mathcal{I} \subseteq \mathtt{D}^\mathcal{I} & \qquad \mathcal{I} \models \mathtt{C}(\mathtt{a}) \quad \text{si } \mathtt{a}^\mathcal{I} \in \mathtt{C}^\mathcal{I} \\ \mathcal{I} \models \mathtt{C} \equiv \mathtt{D} & \text{si } \mathtt{C}^\mathcal{I} = \mathtt{D}^\mathcal{I} & \qquad \mathcal{I} \models \mathtt{r}(\mathtt{a}, \mathtt{b}) \quad \text{si } (\mathtt{a}^\mathcal{I}, \mathtt{b}^\mathcal{I}) \in \mathtt{r}^\mathcal{I} \end{array}$$

Étant donné une base de connaissances $\mathtt{BC}$ et une interprétation $\mathcal{I}$, $\mathcal{I}$ satisfait $\mathtt{BC}$ – dénoté par $\mathcal{I} \models \mathtt{BC}$ – si $\mathcal{I} \models f$ pour chaque $f \in \mathtt{BC}$. Un *modèle* de $\mathtt{BC}$ est une interprétation satisfaisant $\mathtt{BC}$. Une base de connaissances $\mathtt{BC}$ entraîne une formule f – dénoté par $\mathtt{BC} \models f$ – si tout modèle de $\mathtt{BC}$ est un modèle de f. Une tautologie est une formule f satisfaite par toute interprétation. « f est une tautologie » s'écrit $\models f$. Deux bases de connaissances sont dites équivalentes si chaque modèle de l'une d'elles est un modèle de l'autre et vice-versa.

2.3 Inférences

Soit $\mathtt{BC}$ une base de connaissances. Parmi les inférences classiques associées à $\mathcal{ALC}$ il y a les tests de la forme $\mathtt{BC} \models f$, où f est une formule. Par exemple, tester si $\mathtt{BC} \models \mathtt{C} \sqsubseteq \mathtt{D}$ est appelé le *test de subsomption* : il teste si, étant donné $\mathtt{BC}$, le concept $\mathtt{C}$ est plus spécifique que le concept $\mathtt{D}$, ce qui est utile pour organiser les concepts en hiérarchies (p. ex., les hiérarchies d'index de systèmes de RÀPC).

La *classification de concept* consiste, étant donné un concept $\mathtt{C}$, à trouver les concepts atomiques $\mathtt{A}$ apparaissant dans $\mathtt{BC}$ tels que $\mathtt{BC} \models \mathtt{C} \sqsubseteq \mathtt{A}$ (les subsumants de $\mathtt{C}$) et les concepts atomiques $\mathtt{B}$ apparaissant dans $\mathtt{BC}$ tels que $\mathtt{BC} \models \mathtt{B} \sqsubseteq \mathtt{C}$ (les subsumés de $\mathtt{C}$). La *classification d'instance* consiste, étant donné une instance $\mathtt{a}$, à trouver les concepts atomiques $\mathtt{A}$ apparaissant dans $\mathtt{BC}$ tels que $\mathtt{BC} \models \mathtt{A}(\mathtt{a})$. Ces deux inférences peuvent être utilisées pour l'étape de remémoration d'un système de RÀPC.

La *satisfiabilité d'une ABox* consiste à tester si, étant donné $\mathtt{BC}$, une ABox a un modèle. Des inférences importantes peuvent se ramener à cette inférence, p. ex. $\models \mathtt{C} \sqsubseteq \mathtt{D}$ ssi $\{(\mathtt{C} \sqcap \neg \mathtt{D})(\mathtt{a})\}$ est non satisfiable, où $\mathtt{a}$ est une nouvelle instance (n'apparaissant ni dans $\mathtt{C}$, ni dans $\mathtt{BC}$).

D'autres inférences sont définies dans la littérature, mais ne sont pas utiles pour cet article.

2.4 Une procédure de déduction classique pour $\mathcal{ALC}$: la méthode des tableaux

Soit $\mathtt{BC}$ une base de connaissances, $\mathcal{T}_0$, sa TBox, et $\mathcal{A}_0$, sa ABox. La procédure permet de tester si $\mathcal{A}_0$ est satisfiable, étant donné $\mathtt{BC}$.

Prétraitement. La première étape du prétraitement consiste à substituer $\mathcal{T}_0$ par une TBox équivalente $\mathcal{T}'_0$ de la forme $\{\top \sqsubseteq \mathtt{K}\}$, pour un certain concept $\mathtt{K}$. Cela peut être fait tout d'abord en substituant chaque formule $\mathtt{C} \equiv \mathtt{D}$ par deux formules $\mathtt{C} \sqsubseteq \mathtt{D}$ et $\mathtt{D} \sqsubseteq \mathtt{C}$. La TBox résultante est de la forme $\{\mathtt{C}_i \sqsubseteq \mathtt{D}_i\}_{1 \leqslant i \leqslant n}$ et on peut montrer qu'elle équivaut à $\{\top \sqsubseteq \mathtt{K}\}$, avec $\mathtt{K} = (\neg \mathtt{C}_1 \sqcup \mathtt{D}_1) \sqcap \ldots \sqcap (\neg \mathtt{C}_n \sqcup \mathtt{D}_n)$.

La deuxième étape du prétraitement consiste à mettre $\mathcal{T}_0$ et $\mathcal{A}_0$ sous forme normale négative (FNN), ce qui signifie que dans chaque concept apparaissant dans ces bases de connaissances, le signe de négation $\neg$

apparaît uniquement devant des concepts atomiques. Il est toujours possible d'effectuer une telle normalisation en appliquant, tant que c'est possible, les équivalences suivantes (de gauche à droite) :

$$\neg\top \equiv \bot \qquad \neg(\mathtt{C} \sqcap \mathtt{D}) \equiv \neg\mathtt{C} \sqcup \neg\mathtt{D} \qquad \neg\exists \mathtt{r}.\mathtt{C} \equiv \forall \mathtt{r}.\neg\mathtt{C} \qquad \neg\neg\mathtt{C} \equiv \mathtt{C}$$
$$\neg\bot \equiv \top \qquad \neg(\mathtt{C} \sqcup \mathtt{D}) \equiv \neg\mathtt{C} \sqcap \neg\mathtt{D} \qquad \neg\forall \mathtt{r}.\mathtt{C} \equiv \exists \mathtt{r}.\neg\mathtt{C} \qquad \mathtt{C} \sqcup \bot \equiv \mathtt{C}$$

Par exemple, le concept $\neg(\forall \mathtt{r}.(\neg\mathtt{A} \sqcup \exists \mathtt{s}.\mathtt{B}))$ est équivalent au concept $\exists \mathtt{r}.(\mathtt{A} \sqcap \forall \mathtt{s}.\neg\mathtt{B})$, qui est sous FNN.

La TBox de CD donnée dans l'équation (2) est équivalente à la TBox $\{\top \sqsubseteq \mathtt{K}\}$ sous FNN avec

$$\mathtt{K} = (\neg\mathtt{Pomme} \sqcup \mathtt{Piridion}) \sqcap (\neg\mathtt{Poire} \sqcup \mathtt{Piridion}) \sqcap (\neg\mathtt{Piridion} \sqcup \mathtt{Pomme} \sqcup \mathtt{Poire})$$

Processus principal. Étant donné une TBox $\mathcal{T}_0 = \{\top \sqsubseteq \mathtt{K}\}$ et une ABox $\mathcal{A}_0$, toutes deux sous FNN, la méthode des tableaux manipule des ensembles d'ABox, en partant du singleton $\mathcal{D}_0 = \{\mathcal{A}_0^{\mathtt{K}}\}$, avec

$$\mathcal{A}_0^{\mathtt{K}} = \mathcal{A}_0 \cup \{\mathtt{K}(\mathtt{a}) \mid \mathtt{a} \text{ est une instance apparaissant dans } \mathcal{A}_0\}$$

Un tel ensemble d'ABox $\mathcal{D}$ doit être interprété comme une disjonction : $\mathcal{D}$ est satisfiable ssi au moins une $\mathcal{A} \in \mathcal{D}$ est satisfiable.

Chacune des étapes suivantes de l'algorithme consiste à transformer l'ensemble d'ABox courant $\mathcal{D}$ en un autre ensemble d'ABox $\mathcal{D}'$, en appliquant des règles de transformation sur les ABox : quand une règle de transformation ϱ, applicable sur une ABox $\mathcal{A} \in \mathcal{D}$, est sélectionnée par le processus, alors $\mathcal{D}' = (\mathcal{D} \setminus \{\mathcal{A}\}) \cup \{\mathcal{A}^1, \ldots, \mathcal{A}^p\}$ où les $\mathcal{A}^i$ sont obtenues par application de ϱ sur $\mathcal{A}$ (voir plus loin une description de ces règles).

Le processus termine quand aucune règle de transformation n'est applicable.

Une ABox est *fermée* quand elle contient un *conflit* (*clash* en anglais), c.-à-d., une contradiction « évidente ». Pour $\mathcal{ALC}$, les conflits sont les paires d'assertions de la forme $\{\mathtt{A}(\mathtt{a}), (\neg\mathtt{A})(\mathtt{a})\}$.

Par conséquent, une ABox fermée est insatisfiable. Une ABox *ouverte* est une ABox non fermée.

Une ABox est *complète* si aucune règle de transformation ne peut être appliquée sur elle.

Soit $\mathcal{D}_{\text{final}}$, l'ensemble des ABox à la fin du processus, c.-à-d., quand chacune des $\mathcal{A} \in \mathcal{D}$ est complète. Il a été montré (voir, p. ex., [2]) que, avec les règles de transformation présentées ci-dessous, le processus termine toujours, et $\mathcal{A}_0$ est satisfiable étant donné $\mathcal{T}_0$ ssi $\mathcal{D}_{\text{final}}$ contient au moins une ABox ouverte.

Les règles de transformation. Il y a quatre règles de transformation pour la méthode des tableaux appliquée à $\mathcal{ALC}$: $\longrightarrow_\sqcap$, $\longrightarrow_\sqcup$, $\longrightarrow_\forall$, et $\longrightarrow_\exists^{\mathtt{K}}$. Aucune de ces règles n'est applicable sur une ABox fermée. L'ordre de ces règles affecte seulement la performance du système, à l'exception de la règle $\longrightarrow_\exists^{\mathtt{K}}$ qui doit être appliquée seulement quand aucune autre règle n'est applicable sur l'ensemble d'ABox actuel (ceci afin d'assurer la terminaison). Ces règles correspondent à des étapes de déduction au sens où elles ajoutent des assertions déduites d'assertions déjà existantes[3].

La règle $\longrightarrow_\sqcap$ est applicable sur une ABox $\mathcal{A}$ si cette dernière contient une assertion de la forme $(\mathtt{C}_1 \sqcap \ldots \sqcap \mathtt{C}_p)(\mathtt{a})$, et est telle qu'au moins une assertion $\mathtt{C}_k(\mathtt{a})$ $(1 \leqslant k \leqslant p)$ n'appartient pas à $\mathcal{A}$. Cette règle retourne l'ABox $\mathcal{A}'$ définie par

$$\mathcal{A}' = \mathcal{A} \cup \{\mathtt{C}_k(\mathtt{a}) \mid 1 \leqslant k \leqslant p\}$$

La règle $\longrightarrow_\sqcup$ est applicable sur une ABox $\mathcal{A}$ si cette dernière contient une assertion de la forme $(\mathtt{C}_1 \sqcup \ldots \sqcup \mathtt{C}_p)(\mathtt{a})$ mais aucune assertion de la forme $\mathtt{C}_k(\mathtt{a})$ $(1 \leqslant k \leqslant p)$. Cette règle retourne les ABox $\mathcal{A}^1, \ldots, \mathcal{A}^p$ définies, pour $1 \leqslant k \leqslant p$, par

$$\mathcal{A}^k = \mathcal{A} \cup \{\mathtt{C}_k(\mathtt{a})\}$$

La règle $\longrightarrow_\forall$ est applicable sur l'ABox $\mathcal{A}$ si cette dernière contient deux assertions des formes respectives $(\forall \mathtt{r}.\mathtt{C})(\mathtt{a})$ et $\mathtt{r}(\mathtt{a}, \mathtt{b})$ (avec le même $\mathtt{r}$ et le même $\mathtt{a}$), et si $\mathcal{A}$ ne contient pas l'assertion $\mathtt{C}(\mathtt{b})$. Cette règle retourne l'ABox $\mathcal{A}'$ définie par

$$\mathcal{A}' = \mathcal{A} \cup \{\mathtt{C}(\mathtt{b})\}$$

La règle $\longrightarrow_\exists^{\mathtt{K}}$ est applicable sur une ABox $\mathcal{A}$ si

[3] Plus précisément, chacune de ces règles transforme une disjonction d'ABox $\mathcal{D}$ en une autre disjonction d'ABox $\mathcal{D}'$, telle que, étant donné $\mathcal{T}_0$, $\mathcal{D}'$ est satisfiable ssi $\mathcal{D}$ l'est. Cette forme d'équivalence (l'équivalence des satisfiabilités) est celle que l'on retrouve dans la skolémisation des formules de logique du premier ordre (qui remplace les quantificateurs existentiels par des symboles de fonction).

(i) $\mathcal{A}$ contient une assertion de la forme $(\exists \mathtt{r}.\mathtt{C})(\mathtt{a})$;

(ii) $\mathcal{A}$ ne contient pas à la fois une assertion de la forme $\mathtt{r}(\mathtt{a},\mathtt{b})$ et une assertion de la forme $\mathtt{C}(\mathtt{b})$ (avec le même $\mathtt{b}$, et avec les mêmes $\mathtt{C}$ et $\mathtt{a}$ que dans la condition précédente) ;

(iii) Il n'y a aucune instance $\mathtt{c}$ telle que $\{\mathtt{C} \mid \mathtt{C}(\mathtt{a}) \in \mathcal{A}\} \subseteq \{\mathtt{C} \mid \mathtt{C}(\mathtt{c}) \in \mathcal{A}\}$ (cette troisième condition est introduite pour assurer la terminaison de l'algorithme).

Si ces conditions sont remplies, soit $\mathtt{b}$ une nouvelle instance, cette règle retourne

$$\mathcal{A}' = \mathcal{A} \cup \{\mathtt{r}(\mathtt{a},\mathtt{b}), \mathtt{C}(b)\} \cup \{\mathtt{K}(\mathtt{b})\}$$

On peut noter que la TBox $\mathcal{T}_0 = \{\top \sqsubseteq \mathtt{K}\}$ est utilisée ici : étant donné qu'une nouvelle instance $\mathtt{b}$ est introduite, cette instance doit satisfaire la TBox, ce qui correspond à l'assertion $\mathtt{K}(\mathtt{b})$.

Remarque 1 *Après l'application d'une de ces règles sur une ABox de $\mathcal{D}$, la disjonction d'ABox résultante est équivalente à $\mathcal{D}$ (voir note 3).*

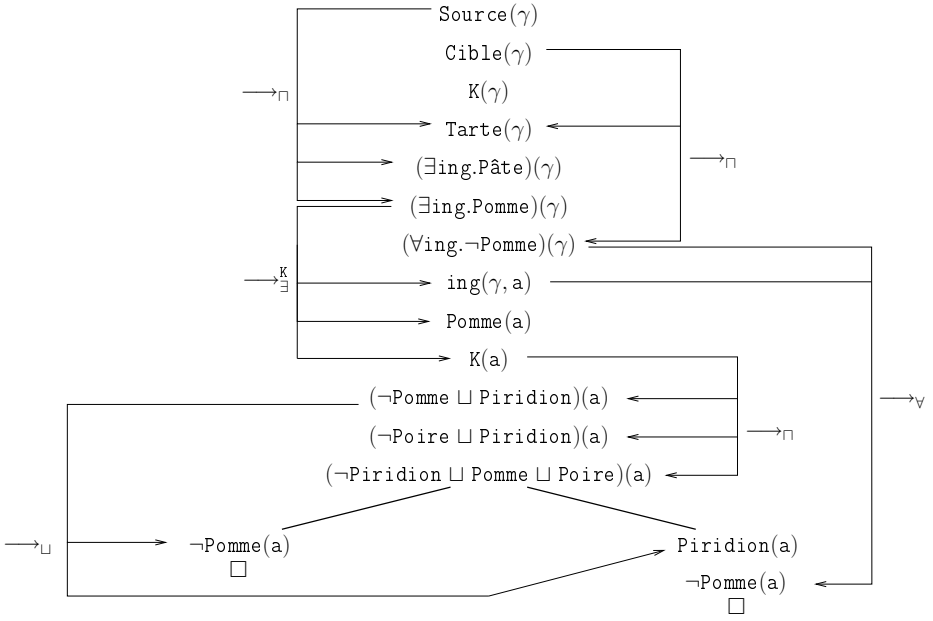


FIG. 1 – Application de la méthode des tableaux montrant que l'ABox $\{\mathtt{Source}(\gamma), \mathtt{Cible}(\gamma)\}$ est non satisfiable, étant donné la TBox $\{\top \sqsubseteq \mathtt{K}\}$ (l'ordre d'application des règles ne respecte pas dans cet exemple la contrainte « $\longrightarrow^{\mathtt{K}}_{\exists}$ doit être appliquée en dernier », cela afin de rendre l'exemple plus simple.).

Exemple. Considérons l'exemple donné à la fin de la section 2.1. Prétendre qu'un cas source représenté par l'instance σ peut être appliqué au cas cible représenté par l'instance γ revient à identifier ces deux instances, p. ex., en substituant σ par γ. Cela conduit à l'ABox $\mathcal{A}_0 = \{\mathtt{Source}(\gamma), \mathtt{Cible}(\gamma)\}$ (avec $\mathtt{Source}$ et $\mathtt{Cible}$ définis dans (4)). La figure 1 représente ce processus. L'arbre en entier représente l'ensemble d'ABox $\mathcal{D}_{\text{final}}$: chacune de ces branches représente une ABox complète $\mathcal{A} \in \mathcal{D}_{\text{final}}$. Au début du processus, les seuls nœuds de cet arbre sont $\mathtt{Source}(\gamma)$, $\mathtt{Cible}(\gamma)$ et $\mathtt{K}(\gamma)$: cela correspond à $\mathcal{D}_0 = \{\mathcal{A}_0^{\mathtt{K}}\}$. Puis, les règles de transformation sont appliquées. On peut noter que seule la règle $\longrightarrow_{\sqcup}$ conduit à la création de plusieurs nœuds-fils d'un nœud. Quand un conflit a été détecté sur une branche (p. ex. $\{\mathtt{Pomme}(\mathtt{a}), (\neg\mathtt{Pomme})(\mathtt{a})\}$) la branche représente une ABox fermée (le conflit est symbolisé par $\square$). Notons que les deux ABox finales sont fermées, ce qui signifie que $\{\mathtt{Source}(\gamma), \mathtt{Cible}(\gamma)\}$ est non satisfiable : le cas source doit être adapté pour pouvoir être réutilisé dans le contexte du cas cible.

3 Un algorithme d'adaptation en $\mathcal{ALC}$

Cette section présente un algorithme d'adaptation en $\mathcal{ALC}$: ses paramètres, son résultat, ses étapes et certaines de ses propriétés.

3.1 Paramètres et résultat de l'algorithme

Les paramètres de l'algorithme sont $\texttt{CD}$, $\mathcal{A}^{\sigma}_{\texttt{srce}}$, $\mathcal{A}^{\gamma}_{\texttt{cible}}$ et $\texttt{coût}$. Son résultat est $\mathcal{D}$.

$\texttt{CD}$ est une base de connaissances en $\mathcal{ALC}$ représentant les connaissances du domaine. Dans l'exemple suivi le long de cet article, son ABox est vide, mais, de façon générale, $\texttt{CD}$ peut contenir des assertions.

Les cas source et cible sont représentés par deux ABox supposées satisfiable étant donné $\texttt{CD}$: $\mathcal{A}^{\sigma}_{\texttt{srce}}$ et $\mathcal{A}^{\gamma}_{\texttt{cible}}$, respectivement. Plus précisément, le cas source est réifié par une instance σ et $\mathcal{A}^{\sigma}_{\texttt{srce}}$ contient les assertions à son sujet. Dans l'exemple ci-dessus, $\mathcal{A}^{\sigma}_{\texttt{srce}}$ contient une seule assertion, $\texttt{Source}(\sigma)$. De façon similaire, le cas cible est réifié par une instance γ et $\mathcal{A}^{\gamma}_{\texttt{cible}}$ contient les assertions concernant γ (une seule assertion dans l'exemple : $\texttt{Cible}(\gamma)$).

Le paramètre $\texttt{coût}$ est une fonction associant à un littéral ℓ une valeur numérique $\texttt{coût}(\ell) > 0$, où un littéral est soit un concept atomique (littéral positif) soit un concept de la forme $\neg\texttt{A}$ où $\texttt{A}$ est atomique (littéral négatif). Intuitivement, plus $\texttt{coût}(\ell)$ est grand, plus il est difficile de renoncer à la vérité d'une assertion $\ell(\texttt{a})$.

L'algorithme retourne $\mathcal{D}$, un ensemble d'ABox $\mathcal{A}$ résolvant le cas cible en adaptant le cas source : $\mathcal{A} \models \mathcal{A}^{\gamma}_{\texttt{cible}}$ et $\mathcal{A}$ réutilise « autant que possible » $\mathcal{A}^{\sigma}_{\texttt{srce}}$. Il peut arriver que $\mathcal{D}$ contienne plusieurs ABox. Dans ce cas, les connaissances du système (en particulier, ceux associés à la fonction $\texttt{coût}$) sont insuffisantes pour faire un choix, et c'est à l'utilisateur de faire ce choix.

3.2 Étapes de l'algorithme

Prétraitement. Soient $\mathcal{T}_{\texttt{CD}}$ et $\mathcal{A}_{\texttt{CD}}$ la TBox et l'ABox de $\texttt{CD}$. Soit $\texttt{K}$ un concept sous forme normale négative (FNN) tel que $\mathcal{T}_{\texttt{CD}}$ soit équivalent à $\{\top \sqsubseteq \texttt{K}\}$. $\mathcal{A}_{\texttt{CD}}$ est simplement ajoutée aux ABox :

$$\mathcal{A}^{\sigma}_{\texttt{srce}} \leftarrow \mathcal{A}^{\sigma}_{\texttt{srce}} \cup \mathcal{A}_{\texttt{CD}} \qquad\qquad \mathcal{A}^{\gamma}_{\texttt{cible}} \leftarrow \mathcal{A}^{\gamma}_{\texttt{cible}} \cup \mathcal{A}_{\texttt{CD}}$$

$\mathcal{A}^{\sigma}_{\texttt{srce}}$ et $\mathcal{A}^{\gamma}_{\texttt{cible}}$ sont ensuite mises sous FNN.

Prétendre que le cas source résout le problème. La réutilisation de $\mathcal{A}^{\sigma}_{\texttt{srce}}$ pour l'instance γ réifiant le cas cible est obtenue en assimilant les instances σ et γ. On obtient alors l'ABox $\mathcal{A}^{\gamma}_{\texttt{srce}}$ par substitution de σ par γ dans $\mathcal{A}^{\sigma}_{\texttt{srce}}$. Soit $\mathcal{A}^{\gamma}_{\texttt{srce,cible}} = \mathcal{A}^{\gamma}_{\texttt{srce}} \cup \mathcal{A}^{\gamma}_{\texttt{cible}}$. Si $\mathcal{A}^{\gamma}_{\texttt{srce,cible}}$ est satisfiable, étant donné $\texttt{CD}$, alors la réutilisation directe du cas source n'entraîne aucune contradiction avec les spécifications du cas cible, il n'apporte que de nouvelles informations à son propos. Par exemple, considérons $\mathcal{A}^{\sigma}_{\texttt{srce}} = \{\texttt{Source}(\sigma)\}$ donnée par l'équation (3), $\mathcal{A}^{\gamma}_{\texttt{cible}} = \{\texttt{Tarte}(\gamma), \texttt{ing}(\gamma, \texttt{p}), \texttt{PâteFeuilletée}(\texttt{p})\}$ (c.-à-d., « Je veux une tarte avec de la pâte feuilletée »), et les connaissances du domaine $\texttt{CD}' = \texttt{CD} \cup \{\texttt{PâteFeuilletée} \sqsubseteq \texttt{Pâte}\}$, avec $\texttt{CD}$ définies dans (2). On peut alors montrer qu'étant donné $\texttt{CD}'$, $\mathcal{A}^{\gamma}_{\texttt{srce,cible}}$ est satisfiable et correspond à une tarte aux pommes avec de la pâte feuilletée.

Il arrive cependant souvent que $\mathcal{A}^{\gamma}_{\texttt{srce,cible}}$ ne soit pas satisfiable, étant donné $\texttt{CD}$. Ceci est vrai pour l'exemple courant. Le principe de l'algorithme d'adaptation consiste alors à réparer $\mathcal{A}^{\gamma}_{\texttt{srce,cible}}$. Par « réparation » de $\mathcal{A}^{\gamma}_{\texttt{srce,cible}}$ nous voulons dire sa modification pour la rendre complète et sans conflit, et donc consistante. La suppression des conflits n'est pas suffisante, les formules dont ces conflits dérivent doivent aussi être supprimées. Cela motive l'introduction dans la section 3.2 des AGraphes qui étendent les ABox en conservant la trace de l'application des règles. De plus, pour obtenir une adaptation plus fine, $\mathcal{A}^{\gamma}_{\texttt{srce}}$ et $\mathcal{A}^{\gamma}_{\texttt{cible}}$ sont complétées par la méthode des tableaux avant d'être combinées.

Appliquer la méthode des tableaux sur $\mathcal{A}^{\gamma}_{\texttt{srce}}$ et sur $\mathcal{A}^{\gamma}_{\texttt{cible}}$, en mémorisant l'application des règles de transformation. L'implantation de cette étape et des suivantes fait intervenir la notion de *graphe assertionnel* (ou *AGraphe*) introduite ici. Un AGraphe $\mathcal{G}$ est un graphe simple dont l'ensemble des nœuds, $\mathit{Nœuds}(\mathcal{G})$, est une ABox, et dont les arcs sont étiquetés par des règles de transformation : si $(\alpha, \beta) \in \mathit{Arcs}(\mathcal{G})$, l'ensemble des arcs orientés, alors $\lambda_{\mathcal{G}}(\alpha, \beta) = \varrho$ indique que β a été obtenu en appliquant ϱ sur α et, peut-être, sur d'autres assertions ($\lambda_{\mathcal{G}}$ est la fonction d'étiquetage du graphe $\mathcal{G}$).

La méthode des tableaux sur les AGraphes repose sur les règles de transformation $\Longrightarrow_{\sqcap}$, $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, et $\Longrightarrow^{\texttt{K}}_{\exists}$. Elles sont similaires aux règles de transformation sur les ABox $\mathcal{ALC}$, à quelques différences près.

La règle $\Longrightarrow_{\sqcap}$ est applicable sur un AGraphe $\mathcal{G}$ si

(i) $\mathcal{G}$ contient un nœud α de la forme $(\mathtt{C}_1 \sqcap \ldots \sqcap \mathtt{C}_p)(\mathtt{a})$;

(ii) $\mathcal{G} \neq \mathcal{G}'$ (c.-à-d., $\mathit{Nœuds}(\mathcal{G}) \neq \mathit{Nœuds}(\mathcal{G}')$ ou $\mathit{Arcs}(\mathcal{G}) \neq \mathit{Arcs}(\mathcal{G}')$) avec $\mathcal{G}'$ défini par

$$\begin{aligned} \mathit{Nœuds}(\mathcal{G}') &= \mathit{Nœuds}(\mathcal{G}) \cup \{\mathtt{C}_k(\mathtt{a}) \mid 1 \leqslant k \leqslant p\} \\ \mathit{Arcs}(\mathcal{G}') &= \mathit{Arcs}(\mathcal{G}) \cup \{(\alpha, \mathtt{C}_k(\mathtt{a})) \mid 1 \leqslant k \leqslant p\} \\ \lambda_{\mathcal{G}'}(\alpha, \mathtt{C}_k(\mathtt{a})) &= \Longrightarrow_{\sqcap} \text{ pour } 1 \leqslant k \leqslant p \\ \lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ pour } e \in \mathit{Arcs}(\mathcal{G}) \end{aligned}$$

Sous ces conditions, l'application de cette règle renvoie $\mathcal{G}'$.

La principale différence entre la règle $\longrightarrow_{\sqcap}$ sur ABox et la règle $\Longrightarrow_{\sqcap}$ sur AGraphes est que cette dernière peut être appliquée à $\alpha = (\mathtt{C}_1 \sqcap \ldots \sqcap \mathtt{C}_p)(\mathtt{a})$ même quand $\mathtt{C}_k(\mathtt{a}) \in \mathit{Nœuds}(\mathcal{G})$ pour tout k, $1 \leqslant k \leqslant p$. Dans ce cas de figure, $\mathit{Nœuds}(\mathcal{G}') = \mathit{Nœuds}(\mathcal{G})$ mais $\mathit{Arcs}(\mathcal{G}') \neq \mathit{Arcs}(\mathcal{G})$: un nouvel arc $(\alpha, \mathtt{C}_k)$ indique ici que $\alpha \models \mathtt{C}_k(\mathtt{a})$ et donc, que si $\mathtt{C}_k(\mathtt{a})$ doit être supprimé, alors α doit aussi être supprimé (voir plus loin, l'étape de réparation de l'algorithme).

De même, les règles $\longrightarrow_{\sqcup}$, $\longrightarrow_{\forall}$, et $\longrightarrow_{\exists}^{\mathtt{K}}$ sont modifiées en $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, et $\Longrightarrow_{\exists}^{\mathtt{K}}$, détaillées dans la figure 2.

On peut appliquer la méthode des tableaux présentée en section 2.4 avec une TBox $\{\top \sqsubseteq \mathtt{K}\}$ et une ABox $\mathcal{A}_0$. Avec comme seule différence que ce sont des AGraphes qui sont manipulés à la place de d'ABox, ce qui entraîne que (1) un AGraphe initial $\mathcal{G}_0^{\mathtt{K}}$ doit être construit à partir de $\mathcal{A}_0^{\mathtt{K}}$ (il est défini par $\mathit{Nœuds}(\mathcal{G}_0^{\mathtt{K}}) = \mathcal{A}_0^{\mathtt{K}}$ et $\mathit{Arcs}(\mathcal{G}_0^{\mathtt{K}}) = \emptyset$), (2) les règles $\Longrightarrow_{\cdot}$ sont utilisées à la place des règles $\longrightarrow_{\cdot}$, et (3) le résultat est un ensemble d'AGraphes complets et ouverts (qui est vide ssi, étant donné $\{\top \sqsubseteq \mathtt{K}\}$, $\mathcal{G}_0^{\mathtt{K}}$ n'est pas satisfiable).

Soient $\{\mathcal{G}_i\}_{1 \leqslant i \leqslant m}$ et $\{\mathcal{H}_j\}_{1 \leqslant j \leqslant n}$ des ensembles d'AGraphes ouverts et complets obtenus par l'application de la méthode des tableaux respectivement sur $\mathcal{A}_0 = \mathcal{A}^{\gamma}_{\mathtt{srce}}$ et $\mathcal{A}_0 = \mathcal{A}^{\gamma}_{\mathtt{cible}}$. Si, étant donné $\{\top \sqsubseteq \mathtt{K}\}$, $\mathcal{A}^{\gamma}_{\mathtt{srce}}$ et $\mathcal{A}^{\gamma}_{\mathtt{cible}}$ sont satisfiables, alors $m \neq 0$ et $n \neq 0$. Si $m = 0$ ou $n = 0$, l'algorithme s'arrête avec la valeur $\mathcal{D} = \{\mathcal{A}^{\gamma}_{\mathtt{cible}}\}$.

Générer des conflits explicites à partir de $\mathcal{G}_i$ et $\mathcal{H}_j$. On considère un nouveau type d'assertion, réifiant la notion de conflit : l'*assertion de conflit* $\Box\pm\mathtt{A}(\mathtt{a})$ réifie le conflit $\{\mathtt{A}(\mathtt{a}), (\neg\mathtt{A})(\mathtt{a})\}$. Elles sont générées par la règle $\Longrightarrow_{\Box}$. Cette règle est applicable sur l'AGraphe $\mathcal{G}$ si

(i) $\mathcal{G}$ contient deux nœuds $\mathtt{A}(\mathtt{a})$ et $(\neg\mathtt{A})(\mathtt{a})$ (avec le même $\mathtt{A}$ et le même $\mathtt{a}$) ;

(ii) $\mathcal{G} \neq \mathcal{G}'$ avec $\mathcal{G}'$ défini par

$$\begin{aligned} \mathit{Nœuds}(\mathcal{G}') &= \mathit{Nœuds}(\mathcal{G}) \cup \{\Box\pm\mathtt{A}(\mathtt{a})\} \\ \mathit{Arcs}(\mathcal{G}') &= \mathit{Arcs}(\mathcal{G}) \cup \{(\mathtt{A}(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a})), ((\neg\mathtt{A})(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a}))\} \\ \lambda_{\mathcal{G}'}(\mathtt{A}(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a})) &= \lambda_{\mathcal{G}'}((\neg\mathtt{A})(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a})) = \Longrightarrow_{\Box} \\ \lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ pour } e \in \mathit{Arcs}(\mathcal{G}) \end{aligned}$$

Sous ces conditions, la règle renvoie $\mathcal{G}'$.

L'étape suivante de l'algorithme consiste à appliquer la méthode des tableau sur chaque $\mathcal{G}_i \cup \mathcal{H}_j$, pour tout i et j, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$, en utilisant les règles de transformation $\Longrightarrow_{\sqcap}$, $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, $\Longrightarrow_{\exists}^{\mathtt{K}}$, et $\Longrightarrow_{\Box}$. À la différence de la méthode des tableaux présentée plus haut où il était inutile d'appliquer les règles sur les ABox fermées (ou les AGraphes fermés), ici, lorsqu'une règle est applicable sur un AGraphe contenant une assertion de conflit, elle est appliquée. Plusieurs conflits peuvent donc être générés dans le même AGraphe.

Remarque 2 *Si une assertion de conflit $\Box\pm A(a)$ est générée, alors ce conflit est la conséquence d'assertions venant à la fois de $\mathcal{G}_i$ et de $\mathcal{H}_j$, autrement, ce conflit aurait été généré à l'étape précédente de l'algorithme (puisque ces deux AGraphes sont ouverts et complets).*

Une condition nécessaire pour appliquer $\Longrightarrow_{\sqcup}$ sur un AGraphe $\mathcal{G}$ est que $\mathcal{G}$ contienne un nœud α de la forme $(\mathtt{C}_1 \sqcup \ldots \sqcup \mathtt{C}_p)(\mathtt{a})$. Si c'est le cas, alors deux conditions peuvent être considérées :

(a) $\mathcal{G}$ ne contient aucune assertion $\mathtt{C}_k(\mathtt{a})$ $(1 \leqslant k \leqslant p)$. Sous cette condition, la règle renvoie les AGraphes $\mathcal{G}^1, \ldots, \mathcal{G}^p$ définis, pour $1 \leqslant k \leqslant p$, par

$$\begin{aligned} \mathit{Nœuds}(\mathcal{G}^k) &= \mathit{Nœuds}(\mathcal{G}) \cup \{\mathtt{C}_k(\mathtt{a})\} \\ \mathit{Arcs}(\mathcal{G}^k) &= \mathit{Arcs}(\mathcal{G}) \cup \{(\alpha, \mathtt{C}_k(\mathtt{a}))\} \\ \lambda_{\mathcal{G}^k}(\alpha, \mathtt{C}_k(\mathtt{a})) &= \Longrightarrow_{\sqcup} \\ \lambda_{\mathcal{G}^k}(e) &= \lambda_{\mathcal{G}}(e) \text{ for } e \in \mathit{Arcs}(\mathcal{G}) \end{aligned}$$

(b) $\mathcal{G}$ contient une ou plusieurs assertion $\beta_k = \mathtt{C}_k(\mathtt{a})$ telle que $(\alpha, \beta_k) \notin \mathit{Arcs}(\mathcal{G})$. Sous cette condition, $\Longrightarrow_{\sqcup}$ renvoie l'AGraphe $\mathcal{G}'$ obtenu en ajoutant cet arc (α, β_k) à $\mathcal{G}$, avec $\lambda_{\mathcal{G}'}(\alpha, \beta_k) = \Longrightarrow_{\sqcup}$.

La règle $\Longrightarrow_{\forall}$ est applicable sur un AGraphe $\mathcal{G}$ si

(i) $\mathcal{G}$ contient un nœud α_1 de la forme $(\forall \mathtt{r}.\mathtt{C})(\mathtt{a})$ et un nœud α_2 de la forme $\mathtt{r}(\mathtt{a}, \mathtt{b})$;

(ii) $\mathcal{G} \neq \mathcal{G}'$ avec $\mathcal{G}'$ défini par

$$\begin{aligned} \mathit{Nœuds}(\mathcal{G}') &= \mathit{Nœuds}(\mathcal{G}) \cup \{\mathtt{C}(\mathtt{b})\} \\ \mathit{Arcs}(\mathcal{G}') &= \mathit{Arcs}(\mathcal{G}) \cup \{(\alpha_1, \mathtt{C}(\mathtt{b})), (\alpha_2, \mathtt{C}(\mathtt{b}))\} \\ \lambda_{\mathcal{G}'}(\alpha_1, \mathtt{C}(\mathtt{b})) &= \lambda_{\mathcal{G}'}(\alpha_2, \mathtt{C}(\mathtt{b})) = \Longrightarrow_{\forall} \\ \lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ pour } e \in \mathit{Arcs}(\mathcal{G}) \end{aligned}$$

Sous ces conditions, la règle renvoie $\mathcal{G}'$.

La règle $\Longrightarrow^{\mathtt{K}}_{\exists}$ est applicable sur un AGraphe $\mathcal{G}$ si

(i) $\mathcal{G}$ contient un nœud α de la forme $(\exists \mathtt{r}.\mathtt{C})(\mathtt{a})$;

(ii) (a) Soit $\mathcal{G}$ ne contient à la fois $\mathtt{r}(\mathtt{a}, \mathtt{b})$ et $\mathtt{C}(\mathtt{b})$ pour aucune instance $\mathtt{b}$;

(b) Soit $\mathcal{G}$ contient deux assertions $\beta_1 = \mathtt{r}(\mathtt{a}, \mathtt{b})$ et $\beta_2 = \mathtt{C}(\mathtt{b})$, telles que $(\alpha, \beta_1) \notin \mathit{Arcs}(\mathcal{G})$ ou $(\alpha, \beta_2) \notin \mathit{Arcs}(\mathcal{G})$;

(iii) Il n'y a pas d'instance $\mathtt{c}$ telle que $\{\mathtt{C} \mid \mathtt{C}(\mathtt{a}) \in \mathit{Nœuds}(\mathcal{G})\} \subseteq \{\mathtt{C} \mid \mathtt{C}(\mathtt{c}) \in \mathit{Nœuds}(\mathcal{G})\}$ (condition du *set-blocking*, servant à assurer la terminaison de l'algorithme).

Si la condition (ii-a) est satisfaite, soit $\mathtt{b}$ une nouvelle instance. La règle renvoie $\mathcal{G}'$ défini par

$$\begin{aligned} \mathit{Nœuds}(\mathcal{G}') &= \mathit{Nœuds}(\mathcal{G}) \cup \{\mathtt{r}(\mathtt{a}, \mathtt{b}), \mathtt{C}(\mathtt{b}), \mathtt{K}(\mathtt{b})\} \\ \mathit{Arcs}(\mathcal{G}') &= \mathit{Arcs}(\mathcal{G}) \cup \{(\alpha, \mathtt{r}(\mathtt{a}, \mathtt{b})), (\alpha, \mathtt{C}(\mathtt{b}))\} \\ \lambda_{\mathcal{G}'}(\alpha, \mathtt{r}(\mathtt{a}, \mathtt{b})) &= \lambda_{\mathcal{G}'}(\alpha, \mathtt{C}(\mathtt{b})) = \Longrightarrow^{\mathtt{K}}_{\exists} \\ \lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ pour } e \in \mathit{Arcs}(\mathcal{G}) \end{aligned}$$

Sous la condition (ii-b), la règle renvoie $\mathcal{G}'$ défini par

$$\begin{aligned} \mathit{Nœuds}(\mathcal{G}') &= \mathit{Nœuds}(\mathcal{G}) \\ \mathit{Arcs}(\mathcal{G}') &= \mathit{Arcs}(\mathcal{G}) \cup \{(\alpha, \beta_1), (\alpha, \beta_2)\} \\ \lambda_{\mathcal{G}'}(\alpha, \beta_1) &= \lambda_{\mathcal{G}'}(\alpha, \beta_2) = \Longrightarrow^{\mathtt{K}}_{\exists} \\ \lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ pour } e \in \mathit{Arcs}(\mathcal{G}) \end{aligned}$$

FIG. 2 – Les règles de transformation $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, et $\Longrightarrow^{\mathtt{K}}_{\exists}$.

Réparer les assertions de conflit. L'étape précédente a produit, pour chaque $\mathcal{G}_i \cup \mathcal{H}_j$, un ensemble non vide $\mathcal{S}_{ij}$ d'AGraphes. L'étape de réparation consiste à réparer chacun de ces AGraphes $\Gamma \in \mathcal{S}_{ij}$ et à ne conserver que ceux qui minimisent le coût de réparation[4]. Soit $\Gamma \in \mathcal{S}_{ij}$. Si Γ ne contient aucune assertion de conflit, ceci implique que $\mathcal{G}_i \cup \mathcal{H}_j$ est satisfiable et donc $\mathcal{A}^{\gamma}_{\texttt{srce,cible}}$ aussi : il n'y a pas d'adaptation nécessaire. Si Γ contient $\delta \geq 1$ assertions de conflit, alors l'une d'entre elles est choisie et sa réparation produit un ensemble d'AGraphes réparés Γ' contenant $\delta - 1$ conflits. La réparation est ensuite reprise sur Γ', jusqu'à ce qu'il n'y ait plus de conflit[5]. Le coût de la réparation globale est la somme des coûts de chaque réparation. Dans la suite, nous verrons comment est réparé un conflit de Γ.

Le principe de la réparation d'un conflit consiste à supprimer des assertions de Γ de sorte à empêcher que les conflit soient regénérées par l'application des règles. Ainsi, la réparation de tous les conflits doit produire des AGraphes satisfiables (ce qui est une conséquence de la complétude de l'algorithme des tableaux sur $\mathcal{ALC}$). Pour cela, on suit le principe suivant, exprimé comme une règle d'inférence :

$$\frac{\varphi \models \beta \qquad \beta \text{ doit être supprimé}}{\varphi \text{ doit être supprimé}} \tag{5}$$

où β est une assertion et φ est un ensemble minimal d'assertions tel que $\varphi \models \beta$ (φ doit être interprété comme la conjonction de ses formules). Supprimer φ revient à oublier une des assertions $\alpha \in \varphi$: lorsque $\text{card}(\varphi) \geq 2$, il y a plusieurs manières de supprimer φ, et donc, plusieurs AGraphes Γ' peuvent être obtenus à partir de Γ. La relation $\models$ reliant φ et β est représentée par les arcs de Γ. Donc, suivant (5), la suppression d'assertions est propagée suivant les arcs (α, β), de β vers α.

Soit $\beta = \square\pm\texttt{A}(\texttt{a})$, le conflit de Γ qu'il faut supprimer. Soient $\alpha^+ = \texttt{A}(\texttt{a})$ et $\alpha^- = \neg\texttt{A}(\texttt{a})$. α^+ ou α^-, l'un des deux au moins, doit être supprimé. $\mathcal{H}_j$ étant un AGraphe ouvert et complet, soit $\alpha^+ \notin \mathcal{H}_j$, soit $\alpha^- \notin \mathcal{H}_j$ (cf. remarque 2). Trois situations sont possibles :

- Si $\alpha^+ \in \mathcal{H}_j$, alors α^+ ne peut être supprimé : c'est une assertion générée à partir de $\mathcal{A}^{\gamma}_{\texttt{cible}}$. Donc α^- doit être supprimée.
- Si $\alpha^- \in \mathcal{H}_j$, alors α^+ doit être supprimé.
- Si $\alpha^+ \notin \mathcal{H}_j$ et $\alpha^- \notin \mathcal{H}_j$, le choix de l'assertion à supprimer repose sur la minimisation du coût. Si $\texttt{coût}(\texttt{A}) < \texttt{coût}(\neg\texttt{A})$, α^+ doit être supprimée. Si $\texttt{coût}(\texttt{A}) > \texttt{coût}(\neg\texttt{A})$, α^- doit être supprimée. Si $\texttt{coût}(\texttt{A}) = \texttt{coût}(\neg\texttt{A})$, deux AGraphes sont générés : l'un en supprimant α^+, l'autre en supprimant α^-.

Si une assertion β doit être supprimée, la propagation de la suppression suivant un arc (α, β) tel que $\lambda_{\mathcal{G}}(\alpha, \beta) \in \{\Longrightarrow_{\sqcap}, \Longrightarrow_{\sqcup}, \Longrightarrow^{\texttt{K}}_{\exists}\}$ consiste à supprimer α (et à propager la suppression depuis α).

Soit β une assertion à supprimer qui a été déduite par la règle $\Longrightarrow_{\forall}$. Il existe alors deux assertions telles que $\lambda_{\mathcal{G}}(\alpha_1, \beta) = \lambda_{\mathcal{G}'}(\alpha_2, \beta) = \Longrightarrow_{\forall}$. Dans cette situation, deux AGraphes sont générés, l'un fondé sur la suppression de α_1, l'autre sur la suppression de α_2 (lorsque α_1 ou α_2 est dans $\mathcal{H}_j$, un seul AGraphe est généré).

À la fin du processus de réparation, on obtient un ensemble non vide $\{\Gamma_k\}_{1 \leqslant k \leqslant p}$ d'AGraphes sans conflit. Seuls ceux dont le coût de réparation est minimal sont gardés. Soit $\mathcal{A}_k = \mathit{Nœuds}(\Gamma_k)$. Le résultat de la réparation est $\mathcal{D} = \{\mathcal{A}_k\}_{1 \leqslant k \leqslant p}$.

Transformer la disjonction d'ABox $\mathcal{D}$. Si $\mathcal{A}, \mathcal{B} \in \mathcal{D}$ vérifient $\mathcal{A} \models \mathcal{B}$, alors les disjonctions d'ABox $\mathcal{D}$ et $\mathcal{D} \setminus \{\mathcal{A}\}$ sont équivalentes. Cela sert à simplifier $\mathcal{D}$ en supprimant de tels $\mathcal{A}$([6]). Après cette simplification, chaque $\mathcal{A} \in \mathcal{D}$ est réécrit pour enlever les instances i introduites par l'algorithme des tableaux. D'abord, les i qui ne sont reliées à aucune instance nommée, ni directement, ni indirectement, par des assertions $\texttt{r}(\texttt{a}, \texttt{b})$ sont supprimées, ce qui signifie que des assertions avec de tels i sont supprimées (cela peut arriver à cause de l'étape de réparation qui « déconnecte » i des instances nommées). Ensuite, une étape de « dé-skolemisation » est effectuée en remplaçant les instances introduites i par des assertions du type $(\exists\texttt{r}.\texttt{C})(\texttt{a})$. Par exemple, l'ensemble

[4] Dans notre prototype ceci a été amélioré en éliminant les réparations en cours dont le coût dépasse le minimum en cours.

[5] Il se peut que des nœuds supplémentaires doivent être supprimés pour assurer la consistance de l'AGraphe réparé. Ils sont déterminés par analyse des *set-bockings* (cf. figure 2, $\Longrightarrow^{\texttt{K}}_{\exists}$, condition (iii)).

[6] Dans nos tests, nous nous sommes servis de conditions nécessaires à $\mathcal{A} \models \mathcal{B}$ portant sur l'inclusion ensembliste, avec ou sans renommage d'instance introduite. Cela a entraîné une réduction conséquente de la taille de $\mathcal{D}$, ce qui incite à penser que l'algorithme présenté ici peut être grandement amélioré, en évitant la génération d'ABox inutiles.

$\{\mathtt{r}(\mathtt{a},\mathtt{i}_1), \mathtt{A}(\mathtt{i}_1), \mathtt{s}(\mathtt{i}_1,\mathtt{i}_2), \neg\mathtt{B}(\mathtt{i}_2)\}$ est remplacé par $\{(\exists \mathtt{r}.(\mathtt{A} \sqcap \exists \mathtt{s}.\neg\mathtt{B}))(\mathtt{a})\}$. L'algorithme renvoie la valeur finale de $\mathcal{D}$.

Exemple. En reprenant l'exemple de la fin de la section 2.1. L'ensemble des étapes de l'algorithme est trop important pour être développé ici, nous ne verrons que les réparations.

Plusieurs AGraphes sont générés et doivent être réparés mais ils partagent tous le même conflit $\square\pm\mathtt{Pomme}(\mathtt{a})$. Il y a deux réparation possibles et l'ensemble $\mathcal{D}$ obtenu ne dépend que des coûts $\mathtt{coût}(\mathtt{Pomme})$ et $\mathtt{coût}(\neg\mathtt{Pomme})$.

Si $\mathtt{coût}(\mathtt{Pomme}) < \mathtt{coût}(\neg\mathtt{Pomme})$ alors $\mathcal{D}$ est équivalente à $\{\{(\mathtt{Tarte} \sqcap \exists\mathtt{ing}.\mathtt{Pâte} \sqcap \exists\mathtt{ing}.\mathtt{Poire})(\gamma)\}\}$. L'adaptation proposée est une tarte aux poires[7].

Si $\mathtt{coût}(\mathtt{Pomme}) \geq \mathtt{coût}(\neg\mathtt{Pomme})$ alors $\mathcal{D}$ est équivalente à $\{\{(\mathtt{Tarte} \sqcap \exists\mathtt{ing}.\mathtt{Pâte})(\gamma)\}\}$. Seule la pâte a été réutilisée du cas source.

3.3 Propriétés de l'algorithme

L'algorithme d'adaptation termine. Cela peut être prouvé en s'appuyant sur la terminaison de l'algorithme de tableau sur les ABox [2] : comme la réparation enlève au moins un nœud des AGraphes à chaque étape, et que ceux-ci sont finis, elle termine également.

Chaque ABox $\mathcal{A} \in \mathcal{D}$ satisfait les contraintes de $\mathtt{Cible} : \mathcal{A} \models \mathcal{A}^{\gamma}_{\mathtt{cible}}$.

Si $\mathcal{A}^{\gamma}_{\mathtt{cible}}$ est satisfiable alors toute $\mathcal{A} \in \mathcal{D}$ est satisfiable. En d'autres termes, à moins que le cas cible soit en contradiction avec les connaissances du domaine, l'adaptation produit un résultat cohérent. Si $\mathcal{A}^{\gamma}_{\mathtt{srce}}$ est non satisfiable alors $\mathcal{D}$ est équivalente à $\{\mathcal{A}^{\gamma}_{\mathtt{cible}}\}$. Cela signifie que si une ABox $\mathcal{A}^{\sigma}_{\mathtt{srce}}$ de la base de cas est dénuée de sens[8], alors $\mathcal{A}^{\gamma}_{\mathtt{cible}}$ n'est pas modifié (on n'infère rien d'un cas source insatisfiable).

Si le cas source est applicable sous des contraintes du cas cible ($\mathcal{A}^{\gamma}_{\mathtt{srce,cible}} = \mathcal{A}^{\gamma}_{\mathtt{srce}} \cup \mathcal{A}^{\gamma}_{\mathtt{cible}}$ est satisfiable) alors $\mathcal{D}$ contient une seule ABox qui est équivalente à $\mathcal{A}^{\gamma}_{\mathtt{srce,cible}}$: le cas source est réutilisé sans modification pour résoudre le cas cible.

L'adaptation présentée ici peut être considérée comme une approche par généralisation et spécialisation de l'adaptation. Les ABox $\mathcal{A} \in \mathcal{D}$ sont obtenues en « généralisant » $\mathcal{A}^{\gamma}_{\mathtt{srce}}$ en $\mathcal{A}'$ (en supprimant des formules : $\mathcal{A}^{\gamma}_{\mathtt{srce}} \models \mathcal{A}'$) puis $\mathcal{A}'$ est « spécialisée » en $\mathcal{A} = \mathcal{A}' \cup \mathcal{A}^{\gamma}_{\mathtt{cible}}$.

4 Discussion et travaux proches

Au-delà d'un processus d'adaptation fondé sur l'appariement ? Il y a deux types d'algorithmes pour les inférences déductives classiques en LD : l'algorithme des tableaux présenté ci-dessus et les algorithmes structurels. Le premier est utilisé pour les LD expressives (c.-à-d., pour $\mathcal{ALC}$ et pour les LD étendant $\mathcal{ALC}$). Les autres sont utilisées pour les autres LD (pour lesquelles au moins certaines des inférences déductives sont polynomiales). L'algorithme structurel pour le test de subsomption $\mathtt{BC} \models \mathtt{C} \sqsubseteq \mathtt{D}$ consiste, après une étape de prétraitement, à *apparier* les descripteurs de D avec les descripteurs de C. Cette procédure d'appariement est assez proche de celles utilisées dans la plupart des procédures d'adaptation, de façon explicite ou non (si les cas ont une structure attribut-valeur fixe, en général, les cas source et cible sont appariés attribut par attribut, et le processus d'appariement n'a pas besoin d'être explicite). Les algorithmes structurels apparaissent comme inappropriés pour les LD expressives pour lesquelles la méthode des tableaux est utilisée à la place. L'algorithme d'adaptation présenté dans ce papier, qui s'appuie sur les principes de la méthode des tableaux, n'a pas d'étape d'appariement (même si on peut toujours apparier *a posteriori* les descripteurs du cas source et du cas cible adapté). En partant de ces observations, nous faisons l'hypothèse que, au-delà d'un certain niveau d'expressivité du langage de représentation des connaissances, il devient difficile d'utiliser des techniques d'appariement pour l'adaptation en tenant pleinement compte des connaissances du domaine.

[7] En l'absence de l'axiome $(\mathtt{Pomme} \sqcup \mathtt{Poire} \sqsubseteq \mathtt{Piridion}) \in \mathtt{CD}$, le résultat serait $\mathcal{D} = \{\mathcal{A}\}$ avec $\mathcal{A}$ équivalent à $(\mathtt{Tarte} \sqcap \exists\mathtt{ing}.\mathtt{Pâte} \sqcap \exists\mathtt{ing}.(\mathtt{Piridion} \sqcap \neg\mathtt{Pomme}))(\gamma)$. Autrement dit, l'adaptation consiste à remplacer les pommes par d'autres piridions (des poires, des coings, des nèfles).

[8] Dans un cadre logique, une base de connaissances non satisfiable est équivalente à n'importe quelle base de connaissances insatisfiable et donc, est dénuée de sens.

Autres travaux sur le RÀPC et les LD. Malgré les avantages de l'usage des LD en RÀPC (motivés en introduction), il y a relativement peu de travaux sur le RÀPC et les LD. Dans [10], les concepts d'une LD sont utilisés comme index pour la remémoration d'un planificateur à partir de cas, alors que l'adaptation est effectuée dans un autre formalisme. Dans [14], une LD non expressive est utilisée pour la remémoration et pour l'organisation de la base de cas. Ce travail s'appuie en particulier sur la notion de plus petit subsumant commun (PPSC) pour réifier la similarité entre concepts représentant les cas source et cible : le PPSC des concepts $\mathtt{C}$ et $\mathtt{D}$, quand il existe, est le plus spécifique des subsumants communs à $\mathtt{C}$ et à $\mathtt{D}$ et il met en évidence les caractéristiques communes aux deux concepts. Ainsi, le calcul du PPSC peut être vu comme un algorithme d'appariement, utilisable lors d'un processus d'adaptation. Dans une LD expressive, le PPSC de $\mathtt{C}$ et $\mathtt{D}$ est $\mathtt{C} \sqcup \mathtt{D}$ (ou un concept équivalent) qui n'exprime rien à propos des caractéristiques communes de $\mathtt{C}$ et $\mathtt{D}$.

À notre connaissance, les seules tentatives pour définir un algorithme d'adaptation pour les LD sont [8] et [6]. [8] présente une modélisation du cycle de vie du RÀPC s'appuyant sur les LD. En particulier, il présente une approche d'adaptation par substitution qui consiste à apparier les descripteurs des cas source et cible par une chaîne de rôles (similaire à une chaîne d'assertion $\mathtt{r}(\mathtt{a}_1, \mathtt{a}_2)$, $\mathtt{r}(\mathtt{a}_2, \mathtt{a}_3)$, etc.) afin de mettre en évidence quelles substitutions peuvent être faites. [6] utilise des règles d'adaptation (des reformulations) et une représentation multi-points de vue pour le RÀPC, incluant une étape d'adaptation complexe, alors que l'algorithme présenté ici utilise principalement les connaissances du domaine pour effectuer l'adaptation. Une perspective de recherche sera de voir comment ces approches de l'adaptation peuvent être combinées.

Travaux sur la réparation d'ontologies. Certains travaux sur le web sémantique et la gestion d'ontologies se penchent sur le problème de la réparation d'ontologies et la fusion ou révision d'ontologies contradictoires. On peut trouver une synthèse dans [7]. Il serait intéressant de comparer de façon précise ces travaux avec le nôtre. Notons néanmoins que la plupart de ces travaux, notamment pour des raisons de temps d'exécution, manipulent les ontologies à un niveau syntaxique (suppression ou affaiblissement de formules) alors que notre travail tente de manipuler les connaissances à un niveau sémantique (conséquences de formules, modèles, etc.).

5 Conclusion et perspectives

Cet article présente un algorithme pour l'adaptation dédié aux systèmes de RÀPC dont les cas et les connaissances du domaine sont représentés dans la LD expressive $\mathcal{ALC}$. La première question soulevée par un problème d'adaptation est « Qu'est-ce qui doit être adapté ? » L'algorithme traite cette question d'abord en prétendant que le cas source résout le problème cible, puis en mettant en évidence des inconsistances logiques : ces dernières correspondent aux parties du cas source qui doivent être modifiées afin de répondre à la situation posée par le cas cible. Ces principes sont appliqués à $\mathcal{ALC}$, LD pour laquelle les contradictions sont réifiées par les conflits générés par la méthode des tableaux. La deuxième question soulevée par un problème d'adaptation est « Comment le cas source doit-il être adapté ? » Pour y répondre, l'algorithme répare les contradictions en enlevant (temporairement) des connaissances associées au cas source, jusqu'à ce que la cohérence soit restaurée.

Actuellement, seul un prototype très simple de cet algorithme d'adaptation a été implanté, et il n'est pas très efficace. Une perspective sera de l'implanter efficacement et de façon extensible, en prenant en compte les extensions possibles mentionnées ci-dessous. On peut noter que les travaux sur l'amélioration de la méthode des tableaux pour les LD a conduit à des gains de temps de calcul spectaculaires (voir, en particulier, [9]).

La deuxième perspective sera d'étendre l'algorithme vers d'autres LD expressives. Les inférences déductives pour ces LD sont la plupart du temps implantées grâce à la méthode des tableaux, avec de nouvelles règles de transformation d'ABox et de nouveaux types de conflits. Pour étendre notre approche de l'adaptation, il faut définir de nouvelles règles de transformation sur les AGraphes et les méthodes de réparation des nouveaux types de conflits. En particulier, nous envisageons de l'étendre à $\mathcal{ALC}(\mathtt{D})$, où $\mathtt{D}$ est le domaine concret des n-uplets de nombres (entiers et réels) avec des prédicats de contraintes linéaires. Cela signifie que les cas pourront avoir des attributs numériques et que les connaissances du domaine pourront contenir des contraintes linéaires sur ces attributs. Cette perspective sera aussi une perspective de [4].

Dans notre algorithme, la valeur précise de $\mathtt{coût}(\ell)$, pour un littéral ℓ n'a guère d'importance. Ce qui importe est le signe et la nullité de $\delta(\mathtt{A}) = \mathtt{coût}(\neg\mathtt{A}) - \mathtt{coût}(\mathtt{A})$. $\delta(\mathtt{A}) = 0$ signifie qu'aucune préférence entre

renoncer à $\mathtt{A}(\mathtt{a})$ ou à $\neg\mathtt{A}(\mathtt{a})$ n'est connue du système : celui-ci retournera les deux possibilités et l'utilisateur fera son choix. Ce choix pourra être retenu pour une utilisation future. L'étude détaillée de cet apprentissage interactif est une perspective de ce travail et il pourra s'appuyer sur [5].

L'algorithme d'adaptation présenté ci-dessus peut être considéré comme une approche de l'adaptation par généralisation et spécialisation (cf. section 3.3). En revanche, l'algorithme [6] est une approche de l'adaptation utilisant des règles, une règle (ou reformulation) spécifiant une substitution pertinente pour une certaine classe de cas sources. Une piste pour intégrer ces deux approches est d'utiliser les règles d'adaptation durant le processus de réparation : à la place d'enlever les assertions conduisant à un conflit, une telle règle, si elle est disponible, peut être utilisée pour trouver des substitutions.

Comme cela a été écrit dans l'introduction, ce travail succède à des travaux sur l'adaptation fondée sur la révision des connaissances, bien qu'on ne puisse pas dire que cet algorithme, sous sa forme actuelle, implante un algorithme de révision pour $\mathcal{ALC}$ (p. ex., il ne permet pas la révision d'une TBox par une ABox). Dans [4], l'adaptation fondée sur la révision est généralisée en une combinaison (de plusieurs cas) fondé sur la fusion. Une telle généralisation devrait être applicable à l'algorithme défini dans ce papier : l'ABox $\mathcal{A}^{\gamma}_{\mathtt{srce}}$ est remplacée par plusieurs ABox et les réparations sont appliquées sur ces ABox. Définir précisément cet algorithme et étudier ses propriétés est une autre perspective.

Remerciements. Les auteurs remercient les relecteurs dont les remarques ont permis d'améliorer cet article.

Références

[1] C. E. Alchourrón, P. Gärdenfors et D. Makinson : On the Logic of Theory Change : partial meet functions for contraction and revision. *Journal of Symbolic Logic*, 50:510–530, 1985.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi et P. Patel-Schneider, éditeurs. *The Description Logic Handbook*. Cambridge University Press, cambridge, UK, 2003.

[3] J. Cojan et J. Lieber : Conservative adaptation in metric spaces. *In Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008, Trier, Germany. Proceedings*, pages 135–149, 2008.

[4] J. Cojan et J. Lieber : Belief Merging-based Case Combination. *In* D. C. Wilson et L. McGinty, éditeurs : *8th International Conference on Case-Based Reasoning (ICCBR 2009)*, volume 5650 de *LNAI*, pages 105–119, Seattle, 2009. Springer.

[5] A. Cordier : *Interactive and Opportunistic Knowledge Acquisition in Case-Based Reasoning*. Thèse de doctorat, Université Lyon 1, France, november 2008.

[6] M. d'Aquin, J. Lieber et A. Napoli : Decentralized Case-Based Reasoning for the Semantic Web. *In* Y. Gil et E. Motta, éditeurs : *Proceedings of the 4th International Semantic Web Conference (ISWC 2005)*, LNCS 3729, pages 142–155. Springer, November 2005.

[7] G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis et G. Antoniou : Ontology change : classification and survey. *Knowledge Eng. Review*, 23(2):117–152, 2008.

[8] M. Gómez-Albarrán, P. A. González-Calero, B. Díaz-Agudo et C. Fernández-Conde : Modelling the CBR Life Cycle Using Description Logics. *In* K.-D. Althoff, R. Bergmann et L. K. Branting, éditeurs : *Proc. of the 3rd International Conference on Case-Based Reasoning Research and Development (ICCBR-99)*, LNAI 1650, pages 147–161. Springer, 1999.

[9] I. Horrocks : *Optimising Tableaux Decision Procedures for Description Logics*. Thèse de doctorat, University of Manchester, 1997.

[10] J. Koehler : Planning from Second Principles. *Artificial Intelligence*, 87:145–186, 1996.

[11] J. Kolodner : *Case-Based Reasoning*. Morgan Kaufmann, Inc., 1993.

[12] J. Lieber : Application of the Revision Theory to Adaptation in Case-Based Reasoning : the Conservative Adaptation. *In Proceedings of the 7th International Conference on Case-Based Reasoning (ICCBR-07)*, LNAI 4626, pages 239–253. Springer, Belfast, 2007.

[13] C. K. Riesbeck et R. C. Schank : *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey, 1989.

[14] S. Salotti et V. Ventos : Study and Formalization of a Case-Based Reasoning System Using a Description Logic. *In* B. Smyth et P. Cunningham, éditeurs : *Fourth European Workshop on Case-Based Reasoning, EWCBR-98*, Lecture Notes in Artificial Intelligence 1488, pages 286–297. Springer, 1998.

Raisonnement à partir de cas dynamique pour la réutilisation contextuelle de l'expérience

Amélie Cordier, Bruno Mascret, Alain Mille
Université Lyon 1, LIRIS, UMR5205, F-69622, France
{prenom.nom}@liris.cnrs.fr

Résumé

Cet article traite du raisonnement à partir de l'expérience tracée (RàPET), un paradigme de raisonnement qui exploite les traces d'interaction laissées par les utilisateurs dans des environnements numériques. Dans le RàPET, les traces d'interaction permettent de conserver les expériences de résolution de problème des utilisateurs « en contexte », leur réutilisation s'en trouve donc facilitée. Par ailleurs, les traces d'interaction peuvent être utilisées comme des sources de connaissances pour découvrir d'autres connaissances utiles au processus de raisonnement. Ce papier décrit les principes du RàPET et propose une architecture générale pour le développement des applications à base de traces. Nous mettons en particulier l'accent sur l'articulation entre le RàPET et les systèmes à base de traces (SBT) dans le contexte du développement d'outils d'assistance aux utilisateurs. Pour cela, nous décrivons les connaissances et les modèles de connaissances impliqués dans les tâches de raisonnement. Nous montrons les avantages que présente l'utilisation des traces comme des conteneurs de connaissances pour la réutilisation de l'expérience. Nous discutons également des questions liées à l'ingénierie des connaissances dynamiques et au rôle du RàPET au regard de cette problématique. Enfin, nous présentons un rapide état des lieux des travaux développés dans ce contexte et nous proposons un plan de travail pour le RàPET, afin d'identifier « ce qu'il reste à faire ». Nous montrons également pourquoi le RàPET peut tirer parti de la dynamique actuelle des développements autour des traces tout en apportant des améliorations aux applications à base de traces.

Mots clés : traces, raisonnement à partir de l'expérience tracée, systèmes d'assistance.

1 Introduction

Le travail présenté dans cet article s'intéresse au défi que présente le développement des systèmes dynamiques et réactifs, capables de s'adapter rapidement, et progressivement, aux changements des besoins et des usages de leurs utilisateurs. Un obstacle majeur s'oppose au développement de telles applications : dans la plupart des cas, les besoins et les modifications des usages ne peuvent pas réellement être anticipés. Ceci complique considérablement la mise en œuvre de mécanismes de raisonnement qui pourraient rendre les systèmes plus souples et plus adaptables. Par exemple, dans le contexte du raisonnement à partir de l'expérience, et en particulier en raisonnement à partir de cas, la capacité du système à s'adapter est limitée par le fait que les modèles de connaissances et les mécanismes de raisonnement sont définis lors de l'étape de conception et sont donc très difficiles à faire évoluer.

Ce papier traite donc du raisonnement à partir de l'expérience tracée (RàPET), paradigme de raisonnement qui vise à apporter des solutions plus dynamiques et plus souples à la problématique de la réutilisation de l'expérience. Nous montrons dans la suite pourquoi s'appuyer sur la notion de « trace » permet de proposer des outils ayant ce type de propriétés. Le RàPET repose sur l'exploitation de *traces d'interaction*. En interagissant avec un système quelconque, l'utilisateur laisse dans l'environnement des traces d'interactions qui constituent des inscriptions numériques de ses propres expériences. Nous considérons que les traces d'interaction sont des conteneurs de connaissances dans lesquels les expériences sont stockées implicitement mais ne sont pas réellement organisées à l'avance. Les expériences passées, appelées *épisodes*, sont seulement identifiées et retrouvées lorsqu'un besoin spécifique apparaît. Ce mécanisme garantit la flexibilité et l'adaptabilité du processus, mais il soulève également des problèmes complexes. Comment retrouver une expérience passée dans une trace ? Quid des connaissances dans un système à base de traces ? Sont-elles toutes contenues dans la trace ou bien doit-on s'appuyer également sur d'autres sources de connaissances ? Quels mécanismes de raisonnement doit-on mettre en œuvre ? Comment faire évoluer les connaissances du système ? Comment valider les connaissances ?

Un autre aspect important dans le RàPET est la notion de contexte. En RàPET, les épisodes sont toujours liés aux traces qui les contiennent. Par conséquent, à tout moment, il est possible de retrouver des éléments contextuels liés à l'épisode courant et de les utiliser pour enrichir le processus de raisonnement. Ainsi, le RàPET permet de manipuler et de réutiliser les expériences de façon beaucoup moins contrainte que si elles étaient représentées sous forme de cas statiques et structurés.

Si l'on s'en tient à la définition du RàPC donnée par Schank [1], alors on peut considérer que le RàPET est une forme de RàPC. En revanche, si l'on cherche à se comparer à la définition donnée par Aamodt & Plaza [2], on peut constater quelques différences non négligeables. En apparence, le principe général du RàPET et celui du RàPC94 (celui d'Aamodt & Plaza, très (trop ?) communément accepté) sont très proches : retrouver une expérience passée, puis l'adapter pour apporter une solution au problème courant. Mais en pratique, les mécanismes mis en œuvre diffèrent. En RàPET, on ne peut plus considérer le raisonnement comme un processus cyclique constitué de cinq étapes successives clairement identifiées. Au contraire, les étapes s'enchevêtrent et les allers-retours entre elles se multiplient rapidement pour préciser la définition du problème ainsi que sa résolution (d'où le côté dynamique). L'étape d'élaboration du RàPC prend ici tout son sens : il s'agit d'identifier clairement le problème puis de construire un ensemble d'indices qui permettront de retrouver dynamiquement un épisode similaire dans la trace. La remémoration consiste à retrouver cet épisode, mais elle est rarement immédiate : des négociations sont nécessaires afin de construire le « bon » épisode, en sélectionnant les « bons » éléments dans la trace. C'est ici que la notion de contexte entre en jeu. C'est elle qui permet de décider si un épisode est bon ou non, étant donné le contexte. L'adaptation est également différente, puisque la stratégie d'adaptation ne dépend plus seulement du problème à résoudre, mais aussi du type d'épisode remémoré. En revanche, la mémorisation du problème résolu devient triviale, puisque tout le processus de résolution de problème est lui aussi stocké dans la trace, par définition, et peut donc être ré-exploité de façon transparente par la suite !

Cet article est organisé de la façon suivante. Dans la section 2, nous revenons sur quelques travaux récents qui visent à développer des applications plus dynamiques et plus adaptables. Ces travaux sont majoritairement issus du champ du raisonnement à partir de cas. Dans la section 3, nous traitons du RàPET en commençant par une rapide description du concept de système à base de traces (SBT) puis en donnant quelques exemples. Ensuite, nous exposons les difficultés soulevées par le développement d'applications à base de traces et nous montrons les bénéfices éventuels d'une telle approche dans le contexte des outils d'assistance à l'utilisateur. L'article se termine par une discussion et une rapide présentation des travaux futurs à explorer dans le contexte du raisonnement à partir de l'expérience tracée.

2 Applications dynamiques et évolutives pour la réutilisation de l'expérience

Depuis quelques temps, plusieurs travaux cherchent à repousser les limites du raisonnement à partir de cas afin de développer des applications plus dynamiques, réactives et adaptées aux utilisateurs. Ces travaux partent tous du constat que les outils actuels sont trop « rigides » et ne sont pas capables d'évoluer pour s'adapter à des besoins non-anticipés ou émergeants. Cette rigidité est notamment due à un ensemble de choix effectués au moment de la conception et qui ne peuvent pas être remis en cause ultérieurement. Par exemple, rares sont les applications de RàPC qui permettent de faire évoluer au fil de l'eau la façon de représenter un cas.

Très récemment, ce problème a été soulevé par Susan Craw dans [3]. Dans cet article, elle propose un défi à la communauté, celui de développer le concept de « *Agile CBR* », que l'on pourrait traduire par « raisonnement à partir de cas agile ». L'idée principale du concept d'*Agile CBR* est de transformer l'approche traditionnelle du raisonnement à partir de cas en une approche dynamique, riche en connaissances, auto-organisée et proposant une méthodologie de résolution de problèmes collaborative. L'idée d'*Agile CBR* s'inspire de la programmation agile qui vise, parmi d'autres objectifs, à s'intéresser aux individus et aux interactions ainsi qu'à proposer une réponse rapide au changement. Ainsi, selon Susan Craw, le côté opportuniste du *Agile CBR* et sa flexibilité visent à fournir la capacité nécessaire pour concevoir des processus dynamiques, capables d'exploiter les informations dont ils disposent en temps réel. Le RàPET revendique les mêmes arguments. Les traces constituent une source de connaissances riche qui devrait être particulièrement utile pour traiter des problèmes de gestion des connaissances soulevées par Susan Craw lorsqu'elle parle des défis posés par l'*Agile CBR*. Par exemple, la question de combiner les expériences de plusieurs utilisateurs pour résoudre le problème spécifique d'un utilisateur particulier peut être efficacement traité par un processus de raisonnement qui s'appuie sur la fusion de traces d'interaction individuelles.

La notion de trace peut également être rapprochée de celle de « provenance ». Nous assistons actuellement à un développement des travaux autour de l'idée de provenance des connaissances. Les travaux décrits dans [4] sont une illustration de cette croissance. Le travail décrit dans cet article s'intéresse donc à la provenance des connaissances mais aussi à la façon d'utiliser les informations sur la provenance pour de nombreuses taches (capturer l'expérience, améliorer le calcul de la similarité, maintenir des bases de cas, etc.). Les auteurs montrent que le raisonnement à partir de cas mémorise toujours les expériences passées sous la forme de cas, mais ne retient aucune information au sujet de la provenance des cas. Or, ils montrent également que cette information sur la provenance des cas a souvent une grande valeur. Avec les traces, les informations de provenance sont toujours conservées puisque les traces contiennent toutes les informations collectées. Les cas ne sont retrouvés que lorsqu'un besoin apparait. Ils sont « construits » à la demande, ce qui permet de choisir, en temps réel, toutes les informations qu'ils doivent contenir, y compris les informations de provenance. Nous pensons que les travaux sur les traces et ceux sur la provenance sont très proches et qu'il serait utile de les combiner.

L'idée de prendre en compte le contexte dans le raisonnement n'est pas nouvelle non plus. Par exemple, [5] montre que combiner les méthodologies du raisonnement à partir de cas et la notion de *context awareness* est une façon nouvelle et performante de modéliser et de raisonner à partir d'informations de contexte. Cette contribution montre comment les comportements des utilisateurs peuvent être appris par une approche « raisonnement à partir de cas », et, pour toutes les raisons citées plus haut, nous pensons que cela pourrait être encore plus efficace avec une approche de RàPET. Illustrant cette idée, [6] utilise des logs d'activités avec des marqueurs temporels. Ces données sont nommées « chemins » (*paths*, dans le texte). Les chemins sont collectés par des applications côté client qui sont implémentées dans plusieurs outils (navigateurs Web, éditeurs de textes, etc.). Ces chemins sont ensuite exploités pour rassembler des informations utilisées pour alimenter des systèmes de recommandation.

Ce que nous qualifions d'*épisode* peut être vu comme un « cas historique » tel que décrit dans [7]. Ce travail propose un environnement pour le développement d'applications de raisonnement à partir de cas historique (*Historical Case-Based Reasoning*). Cet environnement permet à la fois l'expression de données temporelles relatives et absolues, représentant les « histoires » du mode réel sous forme de cas. En HCBR, une base de cas est définie formellement comme une collection d'éléments (temporellement indépendants) couplés à leurs références temporelles. Une approche de RàPET embarque déjà tous ces éléments et y ajoute plusieurs autres possibilités, comme l'extension de contexte, la navigation entre les différents niveaux d'abstraction, la détermination de la provenance, l'élaboration dynamique, etc. Elle est donc plus générale.

D'autres travaux tels que [8], relatifs au partage de l'expérience, peuvent être rapprochés du travail décrit ici. Dans HeyStaks (http://www.heystacks.com), les utilisateurs partagent leurs expériences de recherche avec d'autres utilisateurs. Dans ce travail, les auteurs proposent une approche de classification pour améliorer la pertinence d'un *stak* (une liste de résultats de recherche pertinents dans un contexte donné) tout en palliant les problèmes de bruit dans les résultats (spam, erreurs, résultats périmés, etc.). Il est intéressant de noter que dans ce travail, même si les expériences des utilisateurs sont bien collectées, il reste nécessaire d'aider le système à déterminer quel contexte est réellement en cours d'utilisation. Plutôt que d'utiliser uniquement des classifieurs, il pourrait être possible de combiner l'approche avec une approche à base de traces où la trace serait utilisée pour identifier les éléments nécessaires à la détermination du contexte et pourrait aider à la construction de la *stak*.

À titre d'exemple, pour montrer ce que pourrait être un assistant à base de traces utilisant une approche inspirée du raisonnement à partir de cas, nous pouvons citer [9] qui utilise des techniques de raisonnement à partir de cas pour prédire les objectifs des utilisateurs à partir d'une séquence de ses interactions dans un espace de travail. Une approche à base de trace permettrait de généraliser cette aptitude à assister l'utilisateur, quelque soit sa tâche, en fonction de ce que l'ont peut trouver dans les traces d'interaction de ses expériences précédentes.

Les travaux sur le RàPET en sont encore à l'état prospectif, mais ils ouvrent de nombreuses perspectives applicatives. Dans la suite, nous nous appuyons sur les dernières avancées des travaux autour de la théorie de la trace pour proposer une architecture générale de système mettant en œuvre le RàPET. Afin de rendre nos travaux plus explicites, nous considérons le cas applicatif du développement d'une application d'assistance à l'utilisateur. Nous utilisons cette proposition pour identifier les verrous et proposer des perspectives de recherche.

3 Raisonnement à partir de l'expérience tracée

Dans cette section, nous traitons des principes du raisonnement à partir de l'expérience tracée. Pour cela, il est nécessaire d'introduire les concepts nécessaires à la compréhension de ce qu'est « un système à base de traces », c'est-à-dire un système mettant en œuvre tout ou partie des concepts du raisonnement à partir de l'expérience tracée. Afin de présenter ces concepts de façon plus concrète, nous prenons l'exemple applicatif d'un système à base de traces dédié à l'assistance à l'utilisateur. Nous tenons à attirer l'attention du lecteur sur la distinction qui sera faite plus bas entre « le système à base de traces » et « le système de gestion de bases de traces » (dont le nom a été choisi par analogie avec le système de gestion de bases de données). En fin de section, le lecteur trouvera un exemple visant à clarifier un certain nombre de concepts définis ci-après.

Le système à base de traces. Un système à base de traces (SBT) est constitué de trois principaux composants : un système de gestion de bases de traces (SGBT), une ou plusieurs applications observées et une ou plusieurs applications à base de traces. La figure 1 propose une description de l'architecture générale d'un tel système. La figure est simplifiée. Elle présente un cas où l'on ne considère qu'une application observée et qu'une application à base de traces. Elle illustre la question de la conception d'un assistant à base de traces. Le défi ici est de réutiliser l'expérience pour fournir une assistance aux utilisateurs. Pour proposer une assistance efficace, le système peut soit réutiliser l'expérience de l'utilisateur en s'appuyant sur ses propres traces d'interaction, soit réutiliser et adapter des fragments d'expériences d'autres utilisateurs en explorant leurs traces d'interaction.

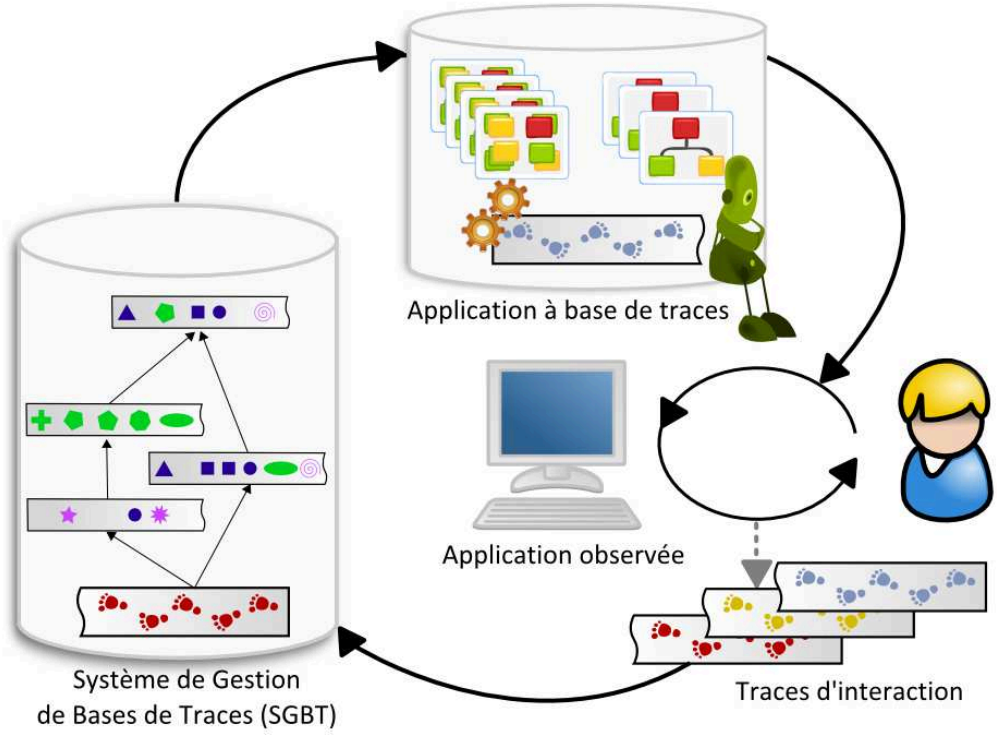


Fig. 1. Architecture générale d'un système à base de traces.

Une façon d'introduire le concept de système de gestion de bases de traces (SGBT) est de faire l'analogie avec un système de fichiers. Le système de fichiers organise, stocke et permet l'accès aux fichiers qu'il contient. Cela dit, sans applications spécifiques pour les lire, les modifier ou les transformer, les fichiers ne présentent que peu d'intérêt pour les utilisateurs qui ne peuvent pas les exploiter directement. Le problème est identique avec les traces. Les traces sont stockées dans le SGBT et si elles ne sont pas

exploitées judicieusement par une application tierce, elles n'ont en général qu'un faible intérêt pour les utilisateurs finaux.

La trace. L'objet central de cette architecture est la trace. Une trace reflète le résultat de l'observation d'une situation donnée. Sur la figure 1, la situation observée est celle de l'utilisation d'une application par un utilisateur (comme c'est souvent le cas). Une trace est un ensemble d'éléments, appelés *obsels* (pour *observed elements*). Une trace modélisée est une trace à laquelle est associé un modèle qui définit formellement la structure et le type des obsels que la trace peut contenir, ainsi que les relations entre ces obsels. Les obsels sont temporellement situés au sein de la trace. Nous pouvons distinguer deux types de traces : les traces premières et les traces transformées. Les traces premières sont le résultat du processus de collecte. Par définition, elles ne sont donc pas transformées. Les traces transformées sont le résultat des opérations de transformation réalisées sur la trace première ou sur une ou plusieurs autres traces transformées. Il existe toujours un lien entre les traces qui ont servi de source à la transformation et la trace résultante (la trace transformée). La figure 2 donne un exemple simple de traces transformées. La trace « du bas » est transformée une première fois, puis cette nouvelle trace est transformée à son tour. Chaque transformation correspond à une abstraction du problème. Le lecteur attentif remarquera que lors du processus d'abstraction, de nouveaux symboles peuvent être introduits afin de représenter des observés (*obsel*) de plus haut niveau. La notion de transformation de traces est détaillée plus bas.

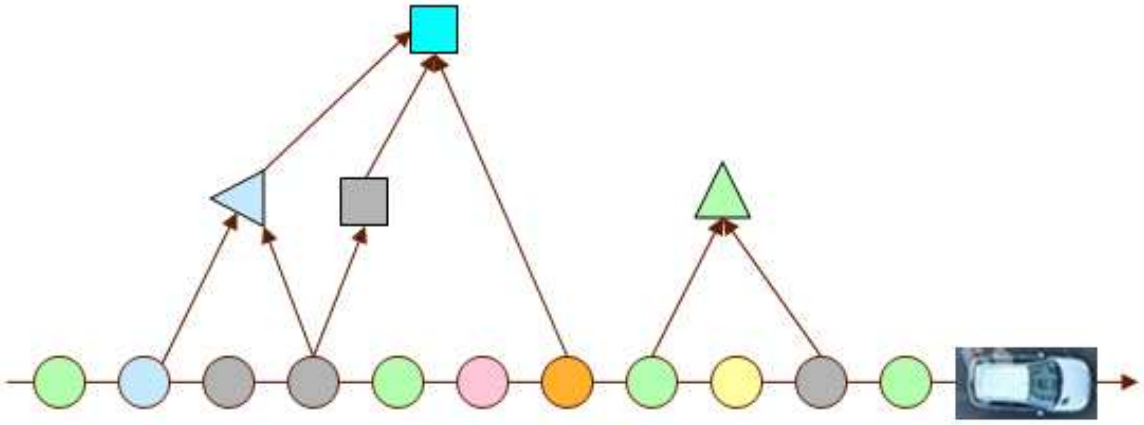


Fig. 2. Une trace et deux niveaux d'abstraction.

Le système de gestion de bases de traces (SGBT). Le SGBT est le composant qui gère les différentes bases de traces. Une base de traces contient à la fois les traces premières ainsi que l'ensemble des traces transformées et leurs modèles associés. Un SGBT fournit des primitives d'entrée/sortie pour manipuler les traces (lecture, écriture, mise à jour, transformations). Les traces sont stockées dans le SGBT au moment du processus de collecte. Ce processus exploite une ou plusieurs sources de collecte qui peuvent être de natures différentes. Soit l'application cible est instrumentée de sorte à ce qu'elle envoie automatiquement au SGBT les éléments à collecter, soit un processus intermédiaire construit la trace première à partir des éléments d'observation disponibles (tels que des fichiers de log des applications observées). Ainsi, une application, un outil, un ensemble de fichiers de logs, une trace d'une autre provenance, etc. constituent autant de sources de collecte. En résumé, le processus de collecte construit la trace primaire qui peut ensuite être transformée en utilisant les opérateurs de transformation existants.

Les transformations de traces. La notion de transformation de traces a une importance capitale dans le RàPET. Une transformation est une opération portant sur une ou plusieurs traces modélisées. Le résultat d'une transformation est une nouvelle trace modélisée, liée aux traces « sources » de la transformation. On peut distinguer plusieurs types de transformations. Nous retiendrons, entre autres, les opérations de filtrage, de fusion et de reformulation. Un opérateur de filtrage, par exemple, produira une nouvelle trace ne contenant que les observés satisfaisant un certain ensemble de critères définis par le filtre. Prenons un exemple très simple dans lequel nous considérons une trace qui contient un ensemble d'observés ayant chacun une propriété « couleur ». Nous définissons un opérateur de filtre imposant de ne garder que les observés dont la couleur est « rouge ». Après application du filtre, la trace transformée ne contiendra que des observés rouges. Cependant, à tout moment, il sera possible de revenir à la trace initiale pour voir, par exemple, quels étaient les voisins immédiats d'un observé donné. Bien entendu, les transformations peuvent être bien plus complexes et peuvent combiner plusieurs opérateurs. Les opérateurs de fusion et de

reformulation fonctionnent de la même façon, mais en plus, ils peuvent être amenés à produire dynamiquement de nouveaux types d'observés pour représenter des observations plus abstraites (comme nous l'avons vu dans l'exemple de la figure 2).

Les opérations de transformation peuvent être manuelles, semi-automatiques ou automatiques. Le SGBT garantit la possibilité, à tout moment, de naviguer entre les traces transformées. En effet, le SGBT conserve en permanence un lien entre la trace transformée et la (ou les) trace(s) source(s) dont elle provient. De plus, des informations sur les opérateurs qui ont permis ces transformations sont également conservées. Autrement dit, le SGBT conserve un treillis de traces transformées afin de garantir la possibilité de navigation. Les traces transformées peuvent être vues comme des représentations différentes d'une même situation à des niveaux d'abstraction différents (cette idée est illustrée dans l'exemple à la fin de cette section). Comme les traces sont liées entre elles, il est toujours possible de se déplacer entre les différents niveaux d'abstraction. Ainsi, on peut considérer un problème à un niveau d'abstraction élevé puis, si cela est nécessaire, redescendre à un niveau d'abstraction plus bas pour obtenir plus de détails. Cette aptitude à naviguer entre les différentes traces modélisées offre donc une flexibilité importante.

Application observée. Le dernier élément de l'architecture est l'outil exploitant les traces. Ce type d'outil peut être connecté au SGBT grâce aux primitives d'entrée/sortie fournies par le SGBT. Les outils peuvent également être connectés avec l'application observée, mais ceci ne rentre pas dans le cadre du travail décrit ici. Ces outils peuvent être de natures diverses : visualisation (interactive) de traces, analyse des usages, assistance à l'utilisateur, etc. Ils doivent s'appuyer sur des connaissances spécifiques pour accomplir certaines tâches (décrites plus bas) en exploitant les traces. Nous appelons ces types de connaissances les connaissances de raisonnement.

Les outils implémentant le paradigme du RàPET doivent donc accomplir plusieurs tâches. La suite de cette section vise à définir ces tâches.

Retrouver un épisode grâce à une signature de tâche. La première tâche consiste à retrouver des épisodes précédents dans la trace en vue de les réutiliser. Pour la recherche d'épisodes, nous nous appuyons sur le concept de signature de tâche. Une signature de tâche contient un ensemble d'éléments distinctifs caractérisant un épisode. À chaque signature de tâche, nous associons un ensemble de mesures de similarité permettant de retrouver dans la trace l'épisode le plus similaire. Nous associons également un ensemble de règles d'adaptation (décrites plus tard). Une autre difficulté vient s'ajouter au problème classique de retrouver des épisodes similaires : la recherche doit s'effectuer parmi les différents niveaux de transformation de la trace. Par conséquent, la recherche doit prendre en compte les différentes représentations possibles d'un même problème. Enfin, cette recherche doit s'effectuer en fonction des informations de contexte disponibles, et ces informations peuvent évoluer au cours du processus. La recherche doit donc être évolutive.

La réutilisation de l'épisode. Une fois retrouvé, l'épisode peut être réutilisé dans beaucoup d'optiques différentes : visualisation, visualisation interactive, rejouage ou adaptation. Dans le cas de la tâche de visualisation, il est nécessaire d'exploiter les différentes connaissances de présentation disponibles de sorte à présenter l'épisode à l'utilisateur de façon claire et compréhensible. La visualisation interactive s'appuie sur le même principe, mais propose en plus à l'utilisateur de naviguer entre les différents niveaux d'abstraction de la trace. La visualisation interactive permet de gérer les différents points de vue sur le même problème de manière efficace et élégante. Pour l'utilisateur, il est possible de considérer le problème à un important niveau d'abstraction puis de redescendre à un niveau plus bas (c'est-à-dire vers une trace transformée plus spécifique) si des détails plus précis sont requis. Le rejouage d'épisodes consiste à retrouver un épisode et à le rejouer à l'identique. Garantir la possibilité de rejouer un épisode à l'identique implique d'importantes contraintes sur la phase de collecte. Notamment, il faut s'assurer que tous les événements nécessaires ont été collectés sous forme d'obels et qu'aucune information ne manquera au moment du rejouage.

L'adaptation d'un épisode. L'adaptation d'un épisode consiste à le réutiliser et à le transformer afin d'apporter une aide à l'utilisateur (assistance, recommandation, exécution automatique d'une tâche, etc.). Dans le contexte du RàPET, il est nécessaire de s'appuyer sur une structure spécifique pour l'adaptation. Cette structure doit offrir un moyen pour le système d'identifier les éléments qui peuvent faire l'objet d'une phase d'adaptation dans l'épisode. Nous nous trouvons ici confrontés à des problèmes similaires à ceux rencontrés en raisonnement à partir de cas et nous avons montré dans un travail précédent [10] qu'il était possible de réutiliser des solutions apportées par les travaux menés dans le contexte du RàPC. En raisonnement à partir de l'expérience tracée, cependant, l'adaptation d'un épisode peut être envisagée à plusieurs niveaux : adaptation de contenu, adaptation de la modalité d'interaction, adaptation de la présentation, navigation entre les différents niveaux d'abstraction (c'est-à-dire adaptation par navigation

entre les traces transformées), etc. Cette diversité parmi les différents types d'adaptation introduit une complexité supplémentaire du point de vue de la représentation des connaissances nécessaires au raisonnement.

Un exemple de RàPET : ABSTRACT. Afin de donner un exemple concret de la façon dont les traces peuvent être collectées, transformées et réutilisées à différents niveaux d'abstraction, nous décrivons brièvement l'application ABSTRACT [11]. ABSTRACT est un projet issu des sciences cognitives et visant à fournir des méthodes et des outils permettant de supporter la découverte de connaissances à partir des traces d'activité. D'un point de vue opérationnel, l'application commence par observer l'activité par différents moyens (collecte de données à partir de différents capteurs, chaque donnée étant marquée par un *timestamp*), ce qui aboutit à la construction de la trace primaire. Par l'intermédiaire des transformations, la trace peut ensuite être manipulée et enrichie avec des symboles plus abstraits. La figure 3 illustre le comportement D'ABSTRACT pour l'analyse du comportement d'un conducteur dans une situation de conduite automobile.

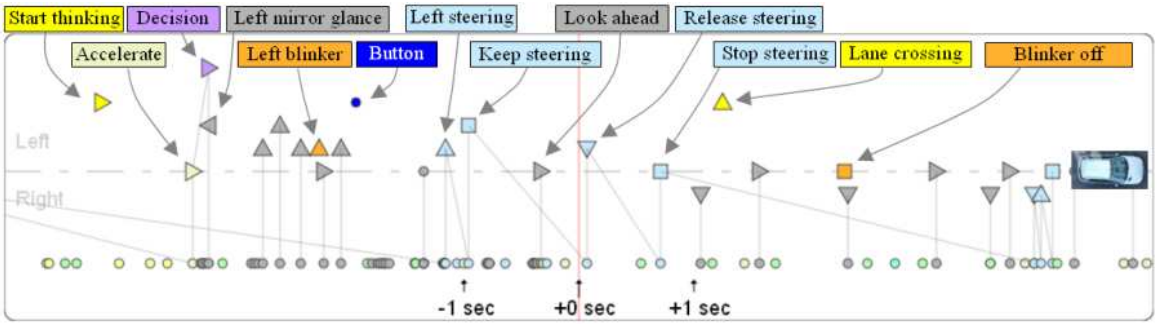


Fig. 3. Transformation des traces dans ABSTRACT.

Résumé. Dans une application à base de traces, le SGBT a un rôle opérationnel : il gère les traces et leurs transformations. Il fournit également les primitives d'entrée/sortie pour manipuler les traces et il permet la navigation entre les différents niveaux de représentation. L'application à base de traces, quant à elle, s'appuie sur des connaissances de présentation et de raisonnement, en plus des traces. Grâce à cela, elle peut atteindre plusieurs objectifs : visualiser un épisode, le réutiliser, l'adapter, etc. Un problème important est celui de l'évolution des connaissances et des modèles de connaissances. L'implémentation de mécanismes pour supporter cette évolution est un problème ouvert. Si la mise à jour de certains conteneurs de connaissances est immédiate (par définition du concept de trace), la mise à jour des modèles de connaissances, en particulier dans le cas des traces modélisées, est plus compliquée et peut nécessiter de faire appel à un processus impliquant un expert. Nous considérons ce problème comme un défi majeur pour le RàPET.

4 Discussion, remarques et conclusion

Le RàPET est un paradigme de résolution de problème dans lequel l'utilisateur et le système collaborent pour résoudre des problèmes mais aussi pour acquérir de nouvelles connaissances. C'est au travers des interactions que l'utilisateur transmet au système les connaissances additionnelles nécessaires pour que celui-ci poursuive son raisonnement. Le système, quant à lui, assiste l'utilisateur en lui apportant des solutions à des problèmes « trop compliqués » pour lui. Au cours de ce processus, la trace constitue un support flexible utilisable à la fois par les humains et par les machines, de façons différentes. Les traces conservent un enregistrement (plus ou moins filtré, en fonction du processus de collecte) des interactions qui ont eu lieu, et donc du processus de résolution de problème. C'est en partie pour cette raison que nous considérons les traces comme une source de connaissances particulièrement riche.

Nous pensons que le raisonnement à partir de l'expérience tracée, parce qu'il offre une approche dynamique et réactive à la question de la collecte et de la réutilisation de l'expérience, est une approche très prometteuse. De plus, les travaux actuels sur les traces sont très actifs et le RàPET pourrait avantageusement tirer profit de cette dynamique. Parmi les recherches sur ce sujet, nous pouvons distinguer deux catégories. La première catégorie concerne les travaux qui traitent des aspects théoriques des traces et de leur gestion (théorie de la trace modélisée, fouille dans les traces pour trouver des motifs pertinents, visualisation

interactive, etc.). La seconde catégorie regroupe les recherches qui visent à produire les applications à base de traces (système à base de traces pour l'analyse de comportement, assistance à l'utilisateur, apprentissage, etc.). Chacun de ces travaux (que nous ne listerons pas ici) a fait l'objet de recherches et d'implémentations plus ou moins indépendantes, mais aucun ne propose actuellement une approche unifiée mettant en œuvre le RàPET comme un élément central permettant à la fois le raisonnement pour la résolution de problèmes, la gestion dynamique des connaissances et la validation des connaissances et du raisonnement. Les réflexions actuelles autour du fameux système à base de traces et du RàPET visent à poser les bases d'une telle approche unifiée.

Parmi les travaux de la première catégorie mentionnée ci-dessus (travaux « théoriques » sur la trace), les travaux traitant des problèmes relatifs aux transformations de traces revêtent une importance toute particulière. En effet, le rôle des transformations est crucial. Les transformations peuvent avoir des objectifs différents : transformer une trace en une trace à un niveau d'abstraction plus élevé, transformer une trace pour en extraire des connaissances ou encore, combiner les deux approches. Cette dernière question soulève un lot de problèmes qui peuvent se résumer aux questions suivantes : est-ce que toutes les connaissances (du point de vue du système) peuvent être représentées sous la forme de transformations, et est-ce que les transformations sont un bon choix pour représenter des connaissances ?

Nous souhaitons également rapprocher les travaux sur les traces et ceux sur la notion de provenance. Nous pensons en effet que ces travaux sont complémentaires. En RàPC, la notion de provenance est utilisée pour améliorer globalement la qualité des résultats produits par le système. Comme nous l'avons montré plus haut, les traces incluent nécessairement ces informations sur la provenance (puisqu'elles gardent une trace de la façon dont les expériences ont été construites). Par conséquent, les informations sur la provenance et l'usage qui en est fait peuvent être remobilisés dans le contexte du RàPET. Les informations sur la provenance peuvent être utilisées de plusieurs façons différentes. Par exemple, dans un système supportant le partage de l'expérience, les informations de provenance peuvent être utilisées pour évaluer la confiance qu'un utilisateur peut avoir en une proposition. La provenance peut aider le système à déterminer quel « type » d'expérience doit être remémorée et réutilisée dans un contexte particulier. Afin d'illustrer ces propos, nous allons détailler l'exemple presque traditionnel du « Copy/Paste » (Copier/Coller, mais la figure est en anglais).

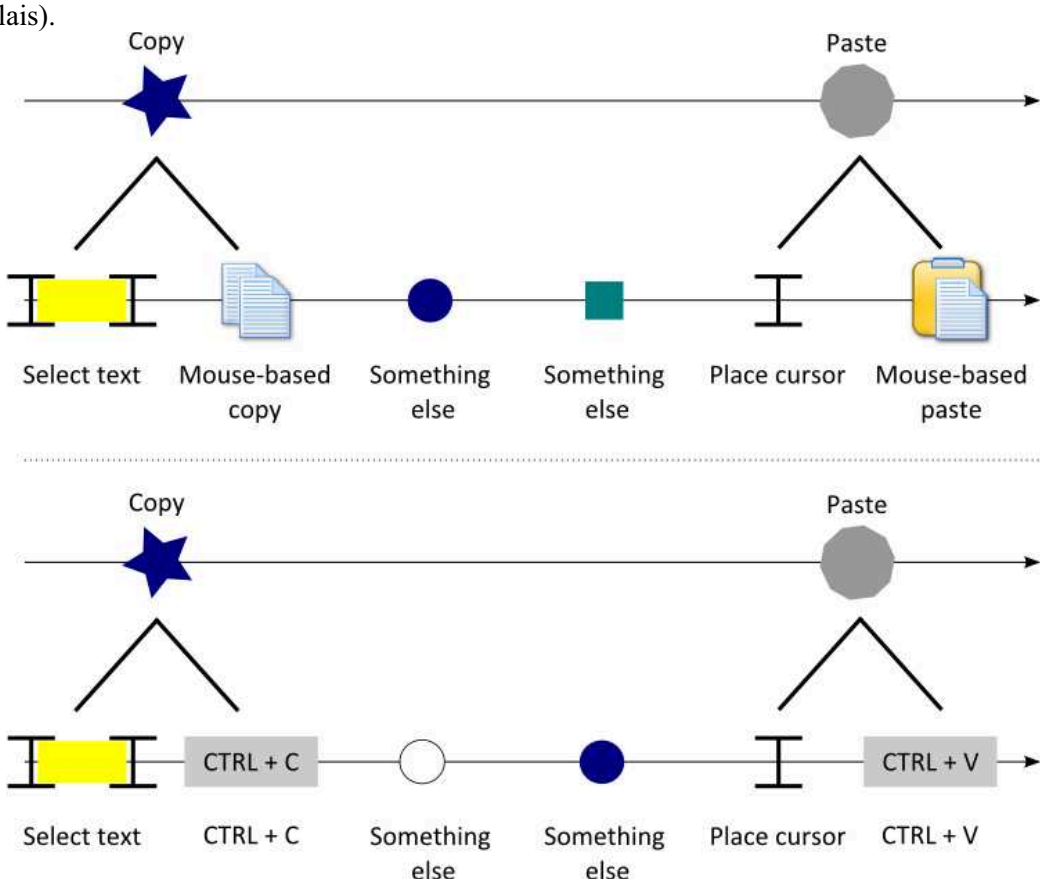


Fig. 4. L'exemple du Copy/Paste.

L'exemple du « Copy/Paste », illustré par la figure 4, montre qu'une même expérience (celle du

Copier/Coller) peut être éprouvée différemment par deux utilisateurs aux usages différents. Supposons qu'un utilisateur aveugle souhaite réutiliser l'expérience de Copier/Coller d'un voyant. Il est raisonnable de penser que la plupart des voyants utilise la souris pour effectuer une telle opération (avec le menu contextuel). Un non-voyant préférera quant à lui n'utiliser que son clavier pour réaliser cette opération. Essayer de partager l'expérience de Copier/Coller entre ces deux utilisateurs en n'utilisant qu'une trace de « bas niveau » est voué à l'échec : les deux utilisateurs ne partagent même pas les mêmes modalités d'interactions avec le système. Il faut donc, dans un premier temps, considérer les traces à un niveau d'abstraction suffisant (c'est-à-dire au niveau ou l'on parle de « Copier/Coller » et non de « Click droit, menu contextuel »). Une fois que le bon niveau d'abstraction est choisi, il s'agit d'adapter la façon d'effectuer l'action. Par exemple, si notre problème est d'aider un non-voyant à faire un Copier/Coller en utilisant comme épisode retrouvé un épisode provenant d'un voyant, il faudra adapter cet épisode en prenant en compte les préférences de l'utilisateur (par exemple, « n'utiliser que le clavier »). Cet exemple soulève également une autre observation que nous trouvons intéressante. Alors que la plupart des utilisateurs semblent préférer la souris pour faire un Copier/Coller (c'est une supposition, surtout pour les besoins de l'exemple), les non-voyants préfèrent probablement une solution « moins commune ». Les informations relatives à la provenance peuvent ici nous aider à trouver les épisodes les mieux adaptés à un utilisateur donné non pas parce qu'ils sont réputés, mais parce que, étant donné le contexte, ils sont plus adaptés. Nous pensons que cette approche est particulièrement importante et prometteuse, en particulier dans le contexte (sans mauvais jeu de mot) des systèmes adaptatifs et des applications d'assistance.

Les perspectives ouvertes par l'implémentation du RàPET sont nombreuses et soulèvent un certain nombre de problèmes. Comment collecter efficacement l'expérience grâce à l'exploitation des traces d'interaction ? Comment définir des mécanismes de transformation pour les traces modélisées et comment faire évoluer ces mécanismes ? Comment adapter les épisodes retrouvés et comment présenter le résultat de cette adaptation aux utilisateurs ? Comment faire évoluer les autres connaissances du système et à partir de quelles sources ? Pour cette dernière question, les traces sont-elles suffisantes ou bien doit-on s'appuyer sur d'autres sources de connaissances, et si oui, comment les intégrer dans le système ?

Toutes ces questions sont autant de pistes de recherche que nous envisageons d'explorer. Nous nous concentrerons en particulier sur le développement de systèmes d'assistance qui soulève des problématiques intéressantes et nombreuses et qui constitue donc un terrain d'expérimentation pertinent.

5 Remerciements

Les auteurs tiennent à remercier les relecteurs de ce papier (ainsi que ceux de la version anglaise) pour leurs commentaires pertinents et leurs questions curieuses (au sens noble). Les éléments de réponses proposés dans ce papier n'étant pas suffisants pour répondre à toutes ces interrogations, nous réfléchissons déjà au prochain article ! Les auteurs remercient également la soigneuse relectrice de cet article.

6 Références

[1] Christopher Riesbeck and Roger Schank. Inside Case-based Reasoning. Northvale, NJ: Erlbaum, 1989.

[2] Agnar Aamodt, and Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. Artificial Intelligence Communications 7, no. 1 (1994): 39-52.

[3] Susan Craw. Agile case-based reasoning: A grand challenge towards opportunistic reasoning from experiences. In Proceedings of the IJCAI-09 Workshop on Grand Challenges in Reasoning from Experiences, pages 33-39, Pasadena, CA, 2009.

[4] David B. Leake, Joseph Kendall-Morwick: Four Heads Are Better than One: Combining Suggestions for Case Adaptation. ICCBR 2009: 165-179.

[5] Andreas Zimmermann. Context-awareness in user modelling: Requirements analysis for a case-based reasoning application. Case-Based Reasoning Research and Development, pages 1064–1064, 2003.

[6] Matthew Chalmers. Abstract paths and contextually specific recommendations. Proc. DELOS/NSF Workshop on Personalization and Recommender Systems in Digital Libraries, Dublin, June 2001.

[7] Jixin Ma and Brian Knight. A framework for historical case-based reasoning. Case- Based Reasoning Research and Development, pages 1067–1067, 2003.

[8] Pierre-Antoine Champin, Peter Briggs, Maurice Coyle and Barry Smyth. Coping with Noisy Search Experiences. In 29th SGAI International Conference on Artificial Intelligence (AI-2009). Springer, pages 5-18, Cambridge, December 2009.

[9] Sven Schwarz and Thomas Roth-Berghofer. Towards goal elicitation by user observation. Workshop on Knowledge and Experience Management at GI FGWM, LLWA 2003.

[10] Amélie Cordier, Bruno Mascret, Alain Mille. Extending Case-Based Reasoning with Traces. In Grand Challenges for reasoning from experiences, Workshop at IJCAI'09, Pasadena, CA. 2009.

[11] Oliver Georgeon, Michael J. Henning, Thierry Bellet, and Alain Mille. Creating Cognitive Models from Activity Analysis: A Knowledge Engineering Approach to Car Driver Modeling. International Conference on Cognitive Modeling, Ann Arbor, MI: Taylor & Francis, pages 43-48.

Améliorer la remémoration par enrichissement de l'ontologie du domaine

Valmi Dufour-Lussier, Jean Lieber, Emmanuel Nauer, Yannick Toussaint
LORIA (CNRS, INRIA, Nancy-Université)
BP 239, 54506 Vandœuvre-lès-Nancy, France
{prénom.nom}@loria.fr

Résumé

Une façon de traiter la remémoration dans les systèmes de raisonnement à partir de cas (RÀPC) est d'exploiter une ontologie pour généraliser progressivement le problème cible, puis d'adapter les cas sources répondant au problème cible généralisé. Cet article montre comment l'enrichissement d'une ontologie améliore la remémoration et, plus globalement, la réponse finale du système. L'ontologie existante est enrichie en ajoutant automatiquement de nouveaux concepts qui vont affiner l'organisation initiale des concepts. Les nouveaux concepts introduits résultent d'un processus de fouille de données — l'analyse de concepts formels (ACF) — sur des données complémentaires relatives aux concepts de l'ontologie, collectées explicitement pour le processus de fouille. Les concepts formels créés par la fouille sont intégrés à l'ontologie, permettant ainsi de généraliser le problème cible de façon plus fine qu'avant cet ajout. Le cadre d'application est le système TAAABLE (`http://taaable.fr`). TAAABLE est un système de RÀPC qui, à partir de contraintes exprimées par un utilisateur, recherche des recettes de cuisine satisfaisant les contraintes ou adapte des recettes de cuisine en proposant de substituer certains ingrédients par d'autres. L'ontologie des ingrédients de TAAABLE a été enrichie à partir de propriétés d'ingrédients collectées dans des données textuelles.

Mots clés : restructuration d'ontologie, analyse de concepts formels, raffinement de la remémoration.

1 Introduction

Nous montrons dans cet article quel est l'impact de l'ontologie dans un système de raisonnement à partir de cas (RÀPC) et comment améliorer la qualité des réponses du système à l'aide d'une ontologie plus fine.

TAAABLE [1], à l'instar de nombreux systèmes de RÀPC, utilise une ontologie de concepts pour remémorer les cas sources les plus *similaires* au cas cible. TAAABLE est un système de recherche et de création de recettes de cuisine par adaptation : étant donné des contraintes énoncées par l'utilisateur telles que la présence ou l'absence d'ingrédients, le type et l'origine du plat souhaité, le moment auquel le plat peut être consommé (au petit-déjeuner, en dessert, etc.) ainsi que sa compatibilité avec certains régimes alimentaires (végétarien, sans alcool, etc.), le système recherche, dans la base de recettes (conçues comme des cas), s'il existe des recettes vérifiant les contraintes. S'il en existe, elles sont proposées à l'utilisateur, sinon le système est capable de retrouver des recettes similaires (i.e. des recettes pour lesquelles les contraintes d'interrogation sont approximativement satisfaites) et de les adapter pour créer de nouvelles recettes. La recherche de recettes similaires se fait à partir de hiérarchies de concepts (d'ingrédients, de type et d'origine des plats) qui permettent de relâcher des contraintes en les généralisant. La relaxation est itérative et progressive : elle conduit à trouver la généralisation la plus spécifique du problème cible (de moindre coût) pour laquelle il existe des recettes dans la base de cas. L'adaptation consiste alors à substituer certains ingrédients des cas sources par ceux souhaités par l'utilisateur.

Les ontologies qui guident ce type de systèmes de RÀPC sont rarement établies spécifiquement pour le processus de remémoration lui-même : soit elles existent déjà et sont injectées dans le système (éventuellement avec certains ajustements), soit elles sont établies pour l'occasion, mais en tant que modélisation des connaissances consensuelles et partageables du domaine d'application (ce qui est le cas de la première version de l'ontologie de TAAABLE). À notre connaissance, dans aucune recherche antérieure elles ne sont établies ni modifiées spécifiquement pour la remémoration : les systèmes de RÀPC les utilisent sans véritablement connaître

ni mesurer l'effet qu'a l'ontologie sur la remémoration. Lors de la généralisation d'un concept C_1 en un concept C, les concepts C_i subsumés par C peuvent être très *similaires* entre eux, ou assez *différents*.

Notre objectif est de mieux contrôler la similarité entre les cas remémorés, lors d'une remémoration par généralisation, et de rendre les généralisations plus progressives. L'idée que nous défendons est que l'introduction de concepts intermédiaires dans l'ontologie rend la généralisation plus fine. En effet, une généralisation qui se faisait initialement en une étape se fera désormais en plusieurs étapes. Remonter d'un niveau dans une hiérarchie plus riche retournera moins de cas, et des cas plus proches entre eux, qu'une remontée dans une hiérarchie moins riche.

Un processus de fouille de données — l'analyse de concepts formels (ACF) — est employé pour dégager de nouveaux concepts qui vont enrichir l'ontologie initiale en affinant l'organisation des concepts. Dans ce but, nous utilisons des sources d'informations complémentaires pour collecter de nouvelles propriétés sur les concepts de l'ontologie initiale. Dans le cadre du système TAAABLE, ce travail est motivé par le fait que certaines adaptations échouent : la recette n'est pas *cuisinable* car certaines actions, appliquées initialement à un ingrédient substitué, ne sont plus applicables à l'ingrédient substituant. Par exemple, lors d'une substitution de la *mozzarella* (fromage relativement ferme) par du *mascarpone* (fromage de consistance crémeuse), il ne sera plus possible de réaliser l'action *couper en tranches* sur l'ingrédient substituant, alors que cette action est courante sur l'ingrédient substitué. Pour réduire cet effet, nous avons travaillé sur l'enrichissement de la hiérarchie des ingrédients, à partir des actions culinaires qui leur sont applicables. Ainsi, la nouvelle proximité des ingrédients dans la hiérarchie prendra en compte les actions qui leur seront potentiellement applicables : la *mozzarella* et le *mascarpone* qui étaient très proches dans l'ontologie initiale seront plus éloignées eu égard aux autres *fromages frais* dont ils font partie.

L'organisation de ce document est la suivante : la section 2 présente le principe du moteur d'inférence de TAAABLE, la section 3 donne un état de l'art sur les approches existantes pour améliorer la remémoration en RÀPC et sur l'utilisation de l'ACF pour la construction et le raffinement d'ontologie, la section 4 détaille le processus de raffinement de l'ontologie, la section 5 montre l'impact de l'ontologie raffinée dans le système TAAABLE.

2 Principe du raisonnement dans TAAABLE

Le raisonnement de TAAABLE est un raisonnement à partir de cas effectuant une remémoration par généralisation de la requête et une adaptation par substitution [1].

2.1 Remémoration

Ontologie du domaine. Une ontologie du domaine guide la remémoration. Cette ontologie est un ensemble de concepts atomiques organisé en une hiérarchie dont le sommet est le concept `Recipe`. Tous les cas de la base de cas sont des instances de ce concept (la base de cas est un livre de recettes). La relation entre deux concepts `A` et `B` de cette ontologie tels que `A` est « en dessous » de `B` dans la hiérarchie est une relation de subsomption : l'ensemble des recettes représentées par `A` est inclus dans l'ensemble des recettes représentées par `B`. Par exemple, le concept `RecipeWithRicotta` désigne le concept des recettes dont un ingrédient est de la ricotta (un fromage frais italien). Ce concept est simplement noté `Ricotta` dans la suite (et dans l'ontologie de TAAABLE), ce qui peut porter à confusion, mais a l'avantage d'être concis. Ainsi, `Riccota` est subsumé par `FreshCheese`, qui est à son tour subsumé par `Cheese`, lui-même subsumé par `Dairy`. Un extrait de la hiérarchie des recettes, selon le point de vue des ingrédients, est donné en Figure 1.

Requête cible. Une requête Q du système TAAABLE est une conjonction de concepts atomiques de l'ontologie. À titre d'exemple, la requête « Je veux une recette avec de la tomate et du fromage frais » s'écrit

1. Dans la troisième version du système, l'adaptation suit toujours ce principe, mais est également étendue par d'autres mécanismes. Par ailleurs, la présentation qui est faite ci-dessous du système TAAABLE (quelle que soit sa version) est simplificatrice à plusieurs égards. Cependant, elle est suffisante pour les besoins de cet article. Pour plus de détails sur le raisonnement dans TAAABLE, nous renvoyons à [7].

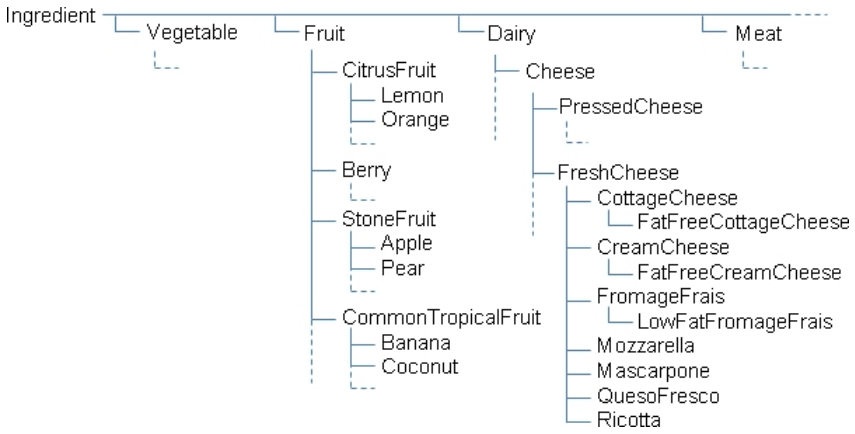


FIGURE 1 – Un extrait de la hiérarchie des (recettes selon le point de vue des) ingrédients, incluant le détail du concept `FreshCheese`.

$Q = \texttt{Tomato} \sqcap \texttt{FreshCheese}$. Si $\texttt{C}$ et $\texttt{D}$ sont deux conjonctions de concepts atomiques (par exemple, deux requêtes), $\texttt{C} = \bigsqcap_i \texttt{A}_i$ est plus spécifique que $\texttt{D} = \bigsqcap_j \texttt{B}_j$ si, pour tout j il existe au moins un i tel que $\texttt{A}_i$ est subsumé par $\texttt{B}_j$ dans l'ontologie.

Cas sources. Les recettes sont représentées, pour le moteur d'inférences, par la conjonction des concepts correspondant aux ingrédients qu'elles possèdent et, donc, dans le même langage que celui des requêtes, Ainsi, une recette avec de la ricotta, de la tomate, de l'huile d'olive, et du sel sera représentée par

$$R = \texttt{Ricotta} \sqcap \texttt{Tomato} \sqcap \texttt{OliveOil} \sqcap \texttt{Salt} \tag{1}$$

Étant donné une recette R et une requête Q, on dit que R résout Q si l'ensemble des recettes ayant la représentation R est contenu dans l'ensemble des recettes représentée par Q. Ainsi, la recette définie par (1) résout la requête $Q = \texttt{OliveOil} \sqcap \texttt{Tomato}$. Cela se calcule simplement en testant si R est plus spécifique que Q. Il est important de noter que le fait que R résout Q ne signifie aucunement qu'il s'agit d'une « bonne » solution, même si on peut partir de l'hypothèse que les recettes de la base de cas sont « bonnes » [2].

Algorithme. Le principe de l'algorithme est de généraliser de façon minimale la requête cible, jusqu'à ce qu'au moins une recette de la base de cas résolve cette requête modifiée. On note $\Gamma_t(Q)$ la requête modifiée à l'instant (l'itération) t et Γ_0 est l'identité. Le coût de la généralisation Γ_t, noté $\texttt{coût}(\Gamma_t)$, croît avec t (voir ci-dessous pour une définition de la fonction $\texttt{coût}$). Chaque généralisation Γ_t s'écrit comme une composition de substitutions $\texttt{A} \rightsquigarrow \texttt{B}$ telles que $(\texttt{A}, \texttt{B})$ est un arc de la hiérarchie représentant l'ontologie. Par ailleurs, cette composition doit être applicable sur Q : si $\Gamma = (\texttt{A}_p \rightsquigarrow \texttt{B}_p) \circ \ldots \circ (\texttt{A}_2 \rightsquigarrow \texttt{B}_2) \circ (\texttt{A}_1 \rightsquigarrow \texttt{B}_1)$ alors $\texttt{A}_1$ est un élément de la conjonction Q, $\texttt{A}_2$ est un élément de la conjonction $(\texttt{A}_1 \rightsquigarrow \texttt{B}_1)(Q)$, etc. L'algorithme de parcours de l'espace des généralisations Γ de Q est un algorithme A* paramétré par la fonction de coût. Il s'arrête quand on trouve au moins une recette R telle que R résout $\Gamma_t(Q)$.

Fonction de coût. Soit Γ une composition de substitutions. La fonction de $\texttt{coût}$ est additive pour la composition : $\texttt{coût}(\sigma_2 \circ \sigma 1) = \texttt{coût}(\sigma_1) + \texttt{coût}(\sigma_2)$. Pour une substitution élémentaire $\sigma = \texttt{A} \rightsquigarrow \texttt{B}$ (où $\texttt{A}$ est subsumé par $\texttt{B}$), le coût est calculé de la façon suivante :

$$\texttt{coût}(\texttt{A} \rightsquigarrow \texttt{B}) = \mu(\texttt{B}) - \mu(\texttt{A}) \quad \text{avec} \quad \mu(\texttt{X}) = \frac{\text{nombre de recettes subsumées par } \texttt{X}}{\text{nombre total de recettes}}$$

On a donc $0 \leqslant \texttt{coût}(\texttt{A} \rightsquigarrow \texttt{B}) \leqslant 1$.

2. Du moins, c'est une hypothèse de travail : si aucune information sur leur qualité n'est associée aux recettes de la base de cas, le plus simple nous semble être de considérer les recettes comme étant bonnes.

	FreshCheese	sliceAble	beatAble	cutAble
Mascarpone	×		×	
Ricotta	×		×	×
Mozzarella	×	×		×
FromageFrais	×		×	
CottageCheese	×		×	
QuesoFresco	×		×	×

TABLE 1 – Un contexte formel de 6 objets (fromages de la catégories des FreshCheese) décrits par 4 propriétés (leur catégorie FreshCheese et 3 actions qui leur sont applicables.

La fonction coût possède la propriété suivante : si Γ est une fonction de généralisation s'appuyant sur une ontologie donnée et qu'on rajoute un nœud dans cette ontologie (mais sans changer la base de recettes) alors le coût de Γ reste inchangé. Par exemple, si $\Gamma = \texttt{Ricotta} \rightsquigarrow \texttt{FreshCheese}$ et qu'on « ajoute » le concept ItalianFreshCheese dans l'ontologie, « entre » Ricotta et FreshCheese, le coût de Γ restera inchangé. Cette propriété montre une différence qui nous semble importante entre cette fonction de coût et celle qui consiste simplement à compter le nombre d'arcs séparant deux nœuds (nombre qui augmente lors d'ajout d'un nœud intermédiaire) ; elle permet d'assurer une certaine stabilité des coûts, indépendamment de l'introduction de concepts intermédiaires.

2.2 Adaptation

Soit Q une requête, Γ une fonction de généralisation de cette requête et R une recette résolvant $\Gamma(Q)$. Soit les $\texttt{C}_i$, les $\texttt{B}_j$ et les $\texttt{S}_k$, concepts atomiques tels que $Q = \bigsqcap_i \texttt{C}_i$ (« C » comme cible), $\Gamma(Q) = \bigsqcap_j \texttt{B}_j$ et $R = \bigsqcap_k \texttt{S}_k$ (« S » comme source). $\Gamma(Q)$ est à la fois plus général que R et que Q, donc pour tout j, il existe k et i tels que $\texttt{S}_k$ et $\texttt{C}_i$ sont subsumés par $\texttt{B}_j$. L'adaptation consiste alors à substituer les $\texttt{S}_k$ par les $\texttt{C}_i$ pour tous les $\texttt{B}_j$ tels que $\texttt{B}_j \neq \texttt{C}_i$([3]). Si plusieurs $\texttt{C}_i$ et/ou $\texttt{S}_k$ correspondent à un seul $\texttt{B}_j$, plusieurs adaptations sont possibles : elles sont toutes présentées à l'utilisateur. Ces substitutions, associées au texte de la recette, indiquent quelles modifications faire sur la recette pour répondre à la requête.

2.3 Exemple

Soit $Q = \texttt{Tomato} \sqcap \texttt{Mascarpone} \sqcap \texttt{Oil}$. Parmi les généralisations de cette requête, supposons que celle de moindre coût (à part l'identité) soit $\Gamma = \texttt{Mascarpone} \rightsquigarrow \texttt{FreshCheese}$. Comme Ricotta est subsumé par FreshCheese dans l'ontologie, TAAABLE retrouve la recette R de (1) et l'adaptation va consister à substituer $\texttt{S}_k = \texttt{Ricotta}$ par $\texttt{C}_i = \texttt{Mascarpone}$ (généralisation-spécialisation passant par $\texttt{B}_j = \texttt{FreshCheese}$).

Cependant, il peut se poser le problème d'une adaptation non réaliste, par exemple demander d'appliquer certaines actions à des objets sur lesquels elles ne sont pas applicables (émietter du mascarpone ou le couper en tranches, par exemple). Nous reviendrons sur cet exemple particulier après avoir introduit les éléments nécessaires à notre approche de raffinement d'ontologie.

3 État de l'art

3.1 Analyse de concepts formels

L'ACF est une approche pour l'analyse de données qui exploite la théorie des treillis. Un *contexte formel* est un triplet $\mathcal{K} = (G, M, I)$, où G est un ensemble d'objets, M un ensemble de propriétés et I la relation sur $G \times M$ exprimant qu'un objet possède une propriété [6]. La Table 1 donne un exemple de contexte : G est un ensemble de 6 objets (qui sont des fromages frais appartenant au concept FreshCheese), M un ensemble de 4 propriétés relatives à ces 6 fromages. Une des propriétés (FreshCheese) vient de la hiérarchie initiale, FreshCheese étant un concept plus générique que les 6 fromages à classifier. Les 3 autres propriétés sont des actions qui

3. Là encore, il y a une simplification : il se peut qu'il y ait plusieurs $\texttt{C}_i$ et/ou $\texttt{S}_k$ qui correspondent à un seul $\texttt{B}_i$. Dans ce cas, plusieurs adaptations sont possibles et présentées à l'utilisateur.

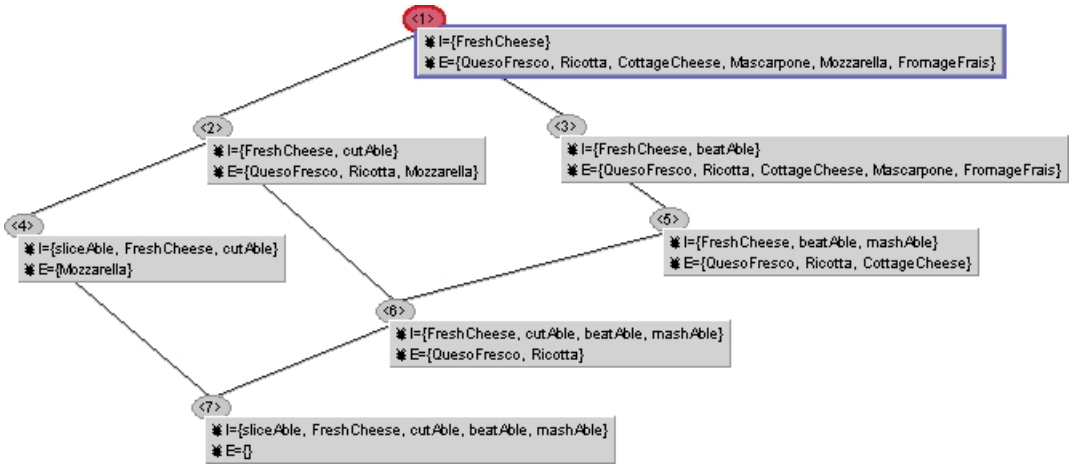


FIGURE 2 – Le treillis de concepts correspondant au contexte formel donné en Table 1.

leur sont potentiellement applicables : `sliceAble` (peut être coupé en tranches), `beatAble` (peut être battu) et `cutAble` (peut être coupé). Cette façon de fusionner, pour un même ensemble d'objets, des propriétés en provenance de plusieurs contextes, est appelée *apposition de contexte*. Formellement, si $\mathcal{K}_1 = (G, M_1, I_1)$ et $\mathcal{K}_2 = (G, M_2, I_2)$ sont deux contextes formels relatifs à un même ensemble d'objets G, un contexte formel, noté $\mathcal{K}_1|\mathcal{K}2$ regroupant $\mathcal{K}_1$ et $\mathcal{K}_2$ peut être construit par : $\mathcal{K}_1|\mathcal{K}_2 = (G, M_1 \cup M_2, I_1 \cup I_2)$ [3].

Un *concept formel* est un couple (I, E), où E est l'ensemble maximal d'objets (appelé *extension*) possédant toutes les propriétés de I, et I est l'ensemble maximal de propriétés (appelé *intension*) partagées par tous les objets de E. Par exemple, $(\{\texttt{FreshCheese}, \texttt{beatAble}, \texttt{cutAble}\}, \{\texttt{QuesoFresco}, \texttt{Ricotta}, \texttt{CottageCheese}\})$ est un concept formel (cf. nœud numéroté 5, Figure 2).

L'ensemble $\mathcal{C}_\mathcal{K}$ de tous les concepts formels du contexte $\mathcal{K} = (G, M, I)$ est partiellement ordonné par l'inclusion des extensions. Cette relation dite de *spécialisation* entre concepts notée $\leq_\mathcal{K}$. $\mathcal{L} = \langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$ est un treillis complet, appelé *treillis de concepts*. Le treillis $\mathcal{L}$ peut être représenté par un diagramme de Hasse dans lequel les nœuds sont les concepts et les arêtes, les liens de spécialisation/généralisation. La Figure 2 illustre le treillis de concepts correspondant au contexte formel donné en Table 1. Le concept le plus général (dans la figure, nœud numéroté 1) contient tous les objets. Dans notre exemple son intension est `FreshCheese`, une propriété partagée par tous les objets. De façon duale, le concept le plus spécifique est défini par l'ensemble de toutes les propriétés. Dans notre exemple, son extension est vide, car aucun objet n'est décrit par toutes les propriétés.

De nombreux algorithmes ont été proposés pour la construction de treillis de concepts (cf. [6]). Pour notre application, nous utilisons la plate-forme CORON (`http://coron.loria.fr/`), qui implémente une large collection d'algorithmes pour la fouille de données symboliques, incluant des algorithmes de construction de treillis de concepts [9]. La Figure 2 provient de GALICIA (`http://www.iro.umontreal.ca/~galicia/`, un autre outil pour l'ACF, qui permet également de les visualiser.

3.2 Construction d'ontologie par ACF

Plusieurs travaux reposent sur l'ACF pour construire des ontologies. L'approche de Cimiano [4] vise à construire une hiérarchie de concepts à partir de textes dans le domaine du tourisme. Les objets (hôtels, voitures, vélos, excursions…) sont caractérisés par les actions qui peuvent leur être appliquées (pouvant être réservé, pouvant être conduit, pouvant être loué…). Afin de réduire la taille du treillis, une première classification numérique est appliquée avant la construction du contexte formel.

Stumme et al. [8] discute de la fusion de deux ontologies. Les classes des ontologies sont repérées dans les textes, et un contexte pour chacune des ontologies est créé dans lequel les classes sont caractérisées par les documents dans lesquels elles apparaissent. L'apposition de contextes est utilisée pour regrouper en fonction de leur citation dans les textes les différentes classes de chacune des deux ontologies.

[3] présente l'ACF comme un cadre unifié pour la construction et l'enrichissement d'ontologies. Il part du constat que, selon les types de ressources textuelles considérées, les informations qui en sont extraites diffèrent. Ainsi, un article scientifique permettra d'identifier des objets et de les associer à des propriétés comme le fait [4]. En revanche, une ontologie organise les concepts d'un domaine en hiérarchie. Cette information d'organisation par spécialisation/généralisation est peu présente dans les articles scientifiques. Dans Bendaoud et *al.* [3], ces deux types d'information sont traités séparément puis, par apposition de contextes, sont combinés pour ne produire qu'un seul et même treillis. Nous utilisons cette méthode pour construire notre ontologie sur les ingrédients en cuisine, exploitant les recettes de cuisine pour caractériser les ingrédients par la façon dont ils sont préparés et exploitant l'ontologie initiale pour introduire les relations de spécialisation/généralisation.

4 Processus de raffinement de l'ontologie

Le processus de raffinement vise à introduire des concepts intermédiaires dans la hiérarchie initiale du système. Pour ce faire, les concepts de la hiérarchie initiale sont caractérisés par des propriétés complémentaires. Ces propriétés permettent de créer un contexte formel qui servira à construire un treillis de concepts par utilisation de l'ACF. Les concepts formels du treillis sont insérés dans la hiérarchie initiale pour enrichir son organisation. Nous détaillons maintenant ces étapes en illustrant nos propos sur le raffinement de la hiérarchie des ingrédients.

4.1 Présentation de la hiérarchie des ingrédients

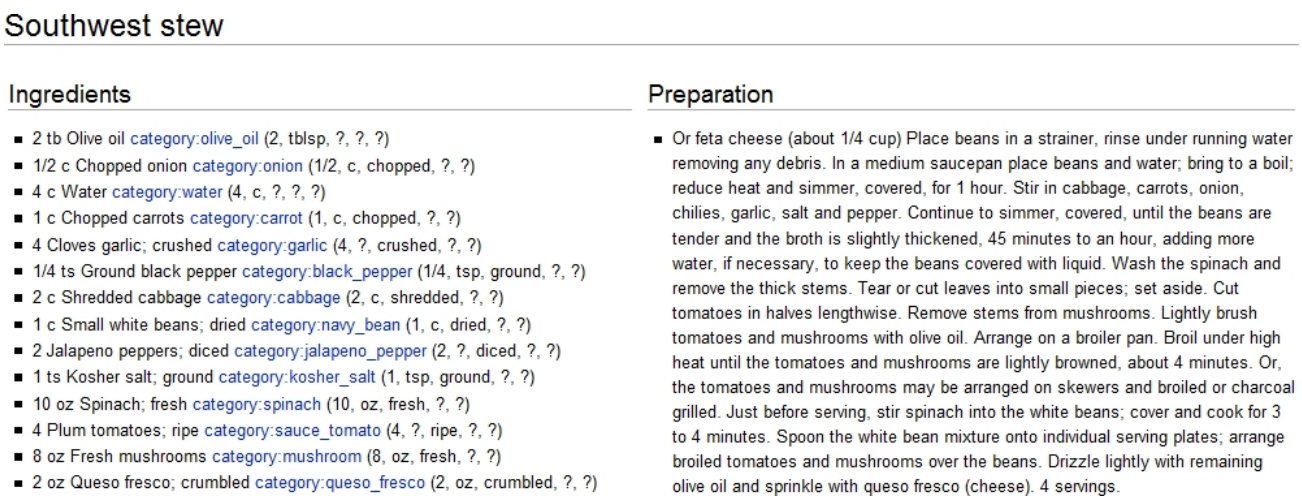
La hiérarchie initiale des ingrédients a été construite semi-automatiquement en partant de ressources existantes du web et des ingrédients présents dans les recettes de la base de cas. Actuellement, la hiérarchie comprend 1450 concepts ; une illustration est donnée en Figure 1. L'accès à l'ontologie (et aux recettes de la base de cas) est géré via un wiki sémantique [2]. La partie *haute* de la hiérarchie contient des concepts très généraux : `Vegetable`, `Fruit` , `Dairy`, `Meat`, ... La partie *basse* de la hiérarchie contient plutôt des concepts qui sont des ingrédients *concrets*, présents dans les recettes.

Dans cette hiérarchie, la *ressemblance* entre *ingrédients-frères* (i.e. sous-concepts directs d'un même concept) n'est pas contrôlée. Par exemple, le mascarpone est aussi proche de la mozzarella que du fromage blanc (`FromageFrais`). Aussi, lors d'une généralisation de requête pour la remémoration, si `Mascarpone` est généralisé en `FreshCheese`, l'ensemble des cas remémorés contiendra indifféremment des recettes avec de la mozzarella et de des recettes avec du mascarpone. C'est pourquoi, nous voulons affiner l'organisation de la hiérarchie pour regrouper entre eux les concepts initiaux se ressemblant (i.e. concepts qui auraient au moins une propriété en commun). La généralisation d'un concept C à un concept parent garantirait que les concepts-frères de C posséderait au moins les mêmes propriétés que C. La généralisation serait alors plus fine.

4.2 Raffinement local

Nous appelons *raffinement local* le fait de raffiner une partie limitée de l'ontologie et non l'ontologie complète : par exemple, la sous-partie relative aux `CommonTropicalFruit` (banane, noix de coco, etc), la sous-partie relative aux `Nut` (noix, noisette, noix de pécan, etc.), la sous-partie relative aux `Berry` (fraise, framboise, myrtille, etc.), la sous-partie relative aux `FreshCheese`, etc. Cette approche a été mise en œuvre pour plusieurs raisons :

- le fait de se focaliser sur une sous-partie de la hiérarchie (i.e. un sous-ensemble de concepts) permet de limiter la taille du contexte en nombre d'objets et en nombre de propriétés associés aux objets. Un contexte limité produit un treillis de moindre complexité : celui-ci est visualisable et interprétable.
- nous ne souhaitons pas construire de concepts qui regroupent des ingrédients n'appartenant pas à une même sous-partie de hiérarchie. Par exemple, nous ne souhaitons pas regrouper les kiwis et les oignons, du simple fait qu'on puisse leur appliquer une action culinaire commune, comme *peler*.
- pour chacune des sous-parties de la hiérarchie, les propriétés prises en compte dans le contexte peuvent varier : pour les noix, *griller* fait sens alors que pour un liquide, *griller* ne fait pas sens.



Southwest stew

Ingredients

- 2 tb Olive oil category:olive_oil (2, tblsp, ?, ?, ?)
- 1/2 c Chopped onion category:onion (1/2, c, chopped, ?, ?)
- 4 c Water category:water (4, c, ?, ?, ?)
- 1 c Chopped carrots category:carrot (1, c, chopped, ?, ?)
- 4 Cloves garlic; crushed category:garlic (4, ?, crushed, ?, ?)
- 1/4 ts Ground black pepper category:black_pepper (1/4, tsp, ground, ?, ?)
- 2 c Shredded cabbage category:cabbage (2, c, shredded, ?, ?)
- 1 c Small white beans; dried category:navy_bean (1, c, dried, ?, ?)
- 2 Jalapeno peppers; diced category:jalapeno_pepper (2, ?, diced, ?, ?)
- 1 ts Kosher salt; ground category:kosher_salt (1, tsp, ground, ?, ?)
- 10 oz Spinach; fresh category:spinach (10, oz, fresh, ?, ?)
- 4 Plum tomatoes; ripe category:sauce_tomato (4, ?, ripe, ?, ?)
- 8 oz Fresh mushrooms category:mushroom (8, oz, fresh, ?, ?)
- 2 oz Queso fresco; crumbled category:queso_fresco (2, oz, crumbled, ?, ?)

Preparation

- Or feta cheese (about 1/4 cup) Place beans in a strainer, rinse under running water removing any debris. In a medium saucepan place beans and water; bring to a boil; reduce heat and simmer, covered, for 1 hour. Stir in cabbage, carrots, onion, chilies, garlic, salt and pepper. Continue to simmer, covered, until the beans are tender and the broth is slightly thickened, 45 minutes to an hour, adding more water, if necessary, to keep the beans covered with liquid. Wash the spinach and remove the thick stems. Tear or cut leaves into small pieces; set aside. Cut tomatoes in halves lengthwise. Remove stems from mushrooms. Lightly brush tomatoes and mushrooms with olive oil. Arrange on a broiler pan. Broil under high heat until the tomatoes and mushrooms are lightly browned, about 4 minutes. Or, the tomatoes and mushrooms may be arranged on skewers and broiled or charcoal grilled. Just before serving, stir spinach into the white beans; cover and cook for 3 to 4 minutes. Spoon the white bean mixture onto individual serving plates; arrange broiled tomatoes and mushrooms over the beans. Drizzle lightly with remaining olive oil and sprinkle with queso fresco (cheese). 4 servings.

FIGURE 3 – Exemple de recette. La recette contient une liste textuelle d'ingrédients et une préparation. Chaque ligne d'ingrédient est analysée pour en extraire des données structurées : quantité, unité, ingrédient et informations complémentaires, tel qu'une action déjà appliquée ou une précision d'état (*fresh* par exemple).

L'idée est donc de se focaliser sur le raffinement de sous-parties (*basses*) de la hiérarchie, en conservant les structurations du *haut*. Les différents raffinements locaux viendront se greffer sur l'ontologie initiale (cf. plus loin).

Formellement, nous notons R le concept racine de la sous-partie de hiérarchie à raffiner. Le contexte $\mathcal{K}_R = (G_R, M_R, I_R)$ dénote le contexte binaire constitué :

- de l'ensemble des objets [4] (des ingrédients) plus spécifiques que R : $G_R = \{x \mid R \sqsupseteq x\}$,
- d'un ensemble de propriétés (des actions culinaires) M_R sélectionnées pour discriminer entre eux les objets de G_R,
- de la relation I_R qui explicite qu'un ingrédient $i \in G_R$ peut subir l'action $a \in M_R$.

Les sections suivantes illustrerons le processus complet sur le raffinement des fromages frais : $R =$ `FreshCheese`.

4.3 Construction du contexte formel pour le raffinement.

Le raffinement d'ontologie consiste à modifier une ontologie existante. Il est nécessaire de garder des propriétés liées à l'ontologie initiale, et d'introduire de nouvelles propriétés pour mieux positionner les concepts (ingrédients) les uns par rapport aux autres. Regrouper le fromage blanc avec le mascarpone et éloigner ces 2 fromages frais de la mozzarella est un résultat attendu. Il s'agit ici d'apposer deux contextes formels :

- un contexte formel en provenance de l'ontologie dans lequel chaque objet-ingrédient est relié à ses concepts plus génériques,
- un contexte formel décrivant des propriétés intrinsèques aux objets-ingrédients. Nous avons choisi de collecter des propriétés relatives aux actions culinaires que les ingrédients peuvent subir : par exemple, le fromage blanc et le mascarpone peuvent être battus tandis que la mozzarella non ; par contre, la mozzarella peut être coupée en tranches, ce qui n'est pas le cas des fromages de consistance crémeuse.

Les propriétés des ingrédients ont été extraits à partir de 73795 recettes de cuisine en provenance de la base de données `Recipe Source` [5]. L'extraction des propriétés est actuellement réalisée depuis la liste des ingrédients qui inclut certaines transformations et du texte de la préparation duquel sont extraits, par un processus de traitement automatique de la langue, les actions associées aux ingrédients. Cette dernière approche, discutée dans [5], repose d'une part sur des méthodes classiques d'analyse morpho-syntaxique, et d'autre part sur un type de représentation du discours spécifiquement conçu pour les textes dits procéduraux (textes d'instructions,

4. Ici, les objets de l'ACF correspondent aux concepts de la hiérarchie des ingrédients, par exemple : `Orange`, `Apple`, etc.
5. `http://www.recipesource.com/`

incluant les recettes) et particulièrement adapté à la résolution de certains phénomènes anaphoriques compliqués trouvés dans ces textes. Un exemple de recette qui est exploité pour l'extraction de propriétés est donné en Figure 3.

L'extraction des propriétés depuis les ingrédients est mise en œuvre dans le processus d'annotation des ingrédients. En effet, les recettes étant initialement sous forme textuelle, un processus d'annotation automatique a été mis en place pour passer à une représentation formelle, manipulable par le moteur de RÀPC. Le processus d'annotation exploite une base terminologique pour identifier dans la partie « Ingrédients » du texte de la recette, les occurrences des noms de concepts de la hiérarchie des ingrédients. Sont également extraits les quantités, les unités et des qualifiants sur les ingrédients tels que *chopped*, *diced*, *crumbled*, etc. (cf. Figure 3). Ces qualifiants retranscrivent des actions culinaires déjà appliquées aux ingrédients. Les couples (ingrédient,qualifiant) ainsi que les couples (ingrédient,action subie) extraits de la préparation textuelle servent de support à la construction du contexte formel. Des illustrations sont visibles dans l'exemple de recette annotée fournie en Figure 3. L'ensemble des propriétés M_R de $\mathcal{K}_R$ contient les actions culinaires que l'on souhaite garder pour raffiner les ingrédients de G_R. Pour $R = \texttt{FreshCheese}$ et l'illustration dans cet article, nous avons retenu $M_R = \{\texttt{mashAble}, \texttt{meltAble}, \texttt{beatAble}, \texttt{mixAble}, \texttt{sliceAble}, \texttt{cutAble}, \texttt{crumbleAble}\}$. La table 2 présente le contexte formel $\mathcal{K}_{\texttt{FreshCheese}}$.

	FreshCheese	CottageCheese	FatFreeCottageCheese	CreamCheese	FatFreeCreamCheese	FromageFrais	LowFatFromageFrais	Mozzarella	Mascarpone	QuesoFresco	Ricotta	mashAble	meltAble	beatAble	mixAble	sliceAble	cutAble	crumbleAble
CottageCheese	×	×										×	×					
FatFreeCottageCheese	×	×	×									×	×					
CreamCheese	×			×								×	×	×	×			
FatFreeCreamCheese	×			×	×							×	×	×	×			
FromageFrais	×					×							×	×	×			
LowFatFromageFrais	×					×	×						×	×	×			
Mozzarella	×							×								×	×	
Mascarpone	×								×				×	×	×			
QuesoFresco	×									×							×	×
Ricotta	×										×		×	×	×			

TABLE 2 – Un contexte formel de 6 objets (fromages de la catégories des `FreshCheese`) décrits par 4 propriétés (leur catégorie `FreshCheese` et 3 actions qui leur sont applicables.

4.4 Intégration d'un treillis local dans la hiérarchie initiale

Les treillis locaux viennent enrichir la hiérarchie initiale. La Figure 4 illustre le treillis construit à partir du contexte $\mathcal{K}_{\texttt{FreshCheese}}$. Le processus d'intégration de chaque treillis local dans la hiérarchie initiale consiste à substituer la sous-partie de hiérarchie enracinée en R par le treillis local *modifié* construit à partir de $\mathcal{K}_R$. La modification du treillis en vue de son intégration dans l'ontologie est réalisée de la façon suivante :

- le concept le plus spécifique est supprimé car son extension est vide (nœud numéro 16 de la Figure 4) ;
- les concepts dont l'extension réduite [6] ne contient qu'un élément sont, par construction du contexte formel, des concepts qui représentent les objets de M_R, l'ensemble des concepts de l'ontologie initiale qui sont à restructurer (nœuds 6, 7, 8, 9, 10, 11, 12, 13, 14 et 15). Ces concepts formels produisent dans l'ontologie un concept dénommé par l'unique élément de l'extension réduite, qui désigne l'objet lui-même. Par exemple, le nœud 8 du treillis donnera naissance au concept `Mozzarella` dans l'ontologie.
- les concepts formels dont l'extension réduite est vide et l'extension est non vide sont les concepts structurants que l'on cherchait à construire. Chacun d'eux donne naissance à un concept de l'ontologie, dénommé par une concaténation des propriétés de l'intension (nœuds 1, 2, 3, 4 et 5). Par exemple, le nœud 5 du treillis (re)donnera naissance au concept `FreshCheese_beatAble_meltable_mixAble` dans l'ontologie.

6. L'extension réduite d'un concept C est l'ensemble des objets qui ne possèdent pas d'autres propriétés que celle contenues dans l'intension de C ; l'extension complète d'un concept C du treillis se retrouve par union des extensions réduites des concepts formels subsumés par C.

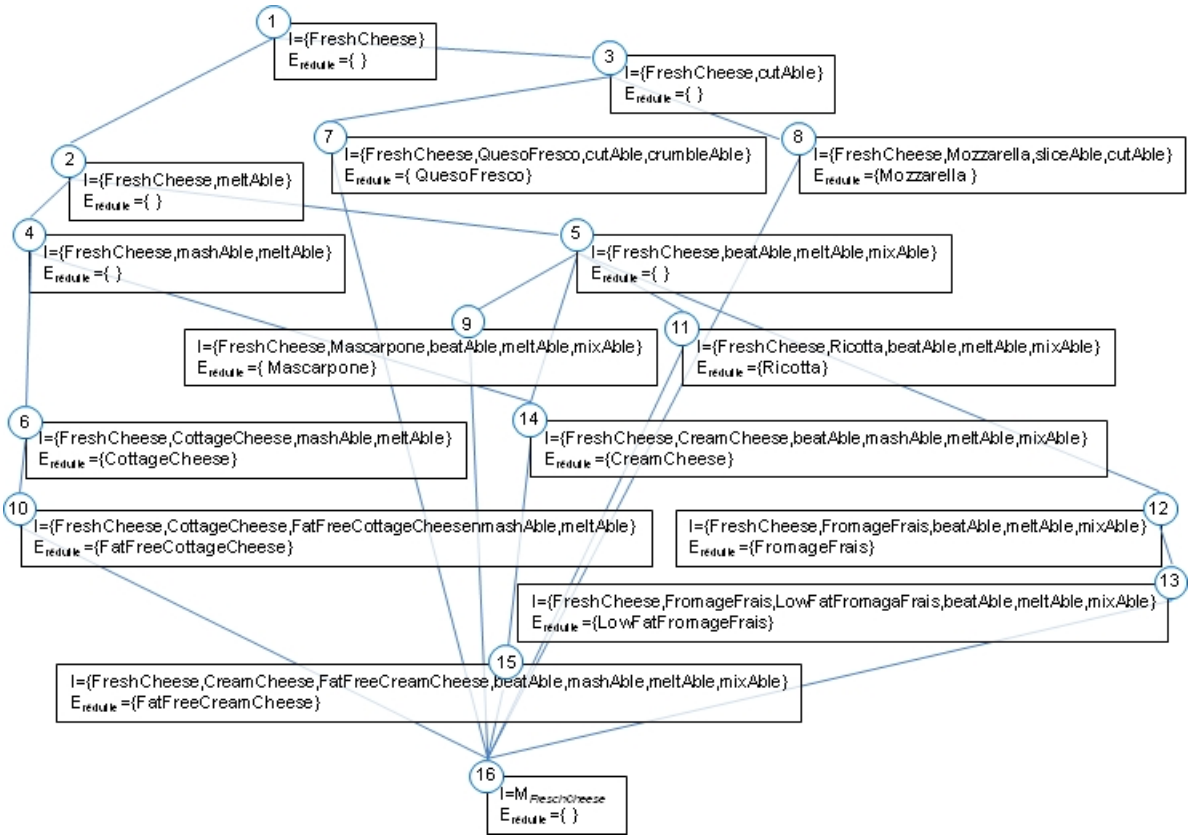


FIGURE 4 – Treillis de concepts restructurant les fromages frais.

La sous-partie de l'ontologie raffinée sur les fromages frais est donnée en partie droite de la figure 5.

5 Expérimentation

Nous illustrons dans cette section l'apport du raffinement de l'ontologie sur le système de RÀPC. Nous donnons un exemple d'utilisation du système TAAABLE *avant* et *après* raffinement pour montrer l'impact du raffinement sur la remémoration.

Pour juger de l'impact du raffinement, nous avons instancié deux systèmes TAAABLE : TAAABLE$_0$ qui fonctionne avec l'ontologie initiale et TAAABLE$_1$ qui fonctionne avec l'ontologie raffinée. Les deux systèmes sont interrogés en parallèle sur une même requête et les résultats sont comparés. La figure 6 donne une illustration de l'interrogation des systèmes sur des recettes avec de la tomate et du mascarpone. Comme dans la base de cas, aucune recette ne contient de la tomate et de la mozzarella, les deux systèmes déclenchent la recherche de cas similaires.

Pour TAAABLE$_0$, la recherche s'arrête après la généralisation `Mascarpone` $\rightsquigarrow$ `FreshCheese`. À cette étape, il existe 8 recettes satisfaisant la requête `Tomato` $\sqcap$ `FreshCheese`, qui sont adaptées en substituant `FreshCheese` par `Mascarpone`. Il existe des recettes contenant simultanément plusieurs `FreshCheese`. Dans ce cas, le système propose de remplacer un ou plusieurs ingrédients (ce qui est le cas des recettes numérotées 1, 3 et 5). Pour TAAABLE$_1$, la remémoration est raffinée et la généralisation s'arrête à un concept plus spécifique que `FreshCheese`, à savoir `FreshCheese_beatAble_meltable_mixAble`, qui correspond au concept des fromages frais qu'on peut battre et mélanger. Ces actions sont également applicables au mascarpone.

L'analyse des résultats présentés en 6 montre que :

– TAAABLE$_1$ ne renvoie que 4 réponses, alors que TAAABLE$_0$ en renvoie 8 (dont les 4 réponses de TAAABLE$_1$, mais avec des substitutions différentes). La généralisation est moins brusque avec l'ontologie raffinée qu'avec l'ontologie initiale.

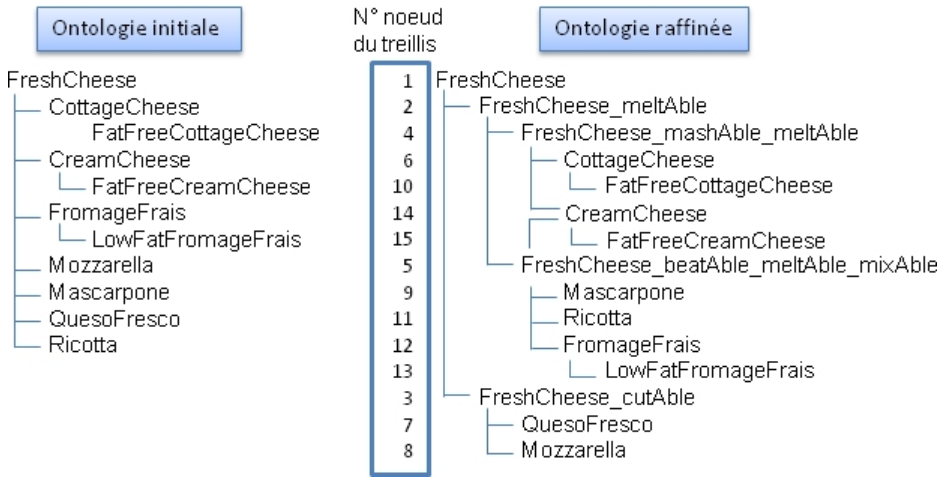


FIGURE 5 – Partie de l'ontologie initiale et de l'ontologie raffinée sur la sous-partie `FreshCheese`.

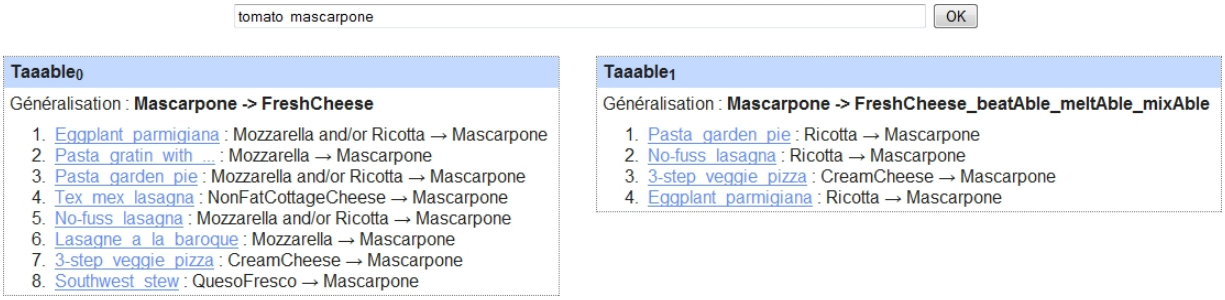


FIGURE 6 – Réponses données par les systèmes TAAABLE$_0$ et TAAABLE$_1$ sur la requête : `Tomato` $\sqcap$ `Mascarpone`.

– les substitutions effectuées par TAAABLE$_0$ sont plus variées que celles effectuées dans TAAABLE$_1$: dans TAAABLE$_1$, le mascarpone vient remplacer de la ricotta ou du *cream cheese* (un fromage crémeux à tartiner), alors que dans TAAABLE$_0$, le mascarpone peut se substituer, en plus, à de la mozzarella, du *non-fat cottage cheese* ou encore du *queso fresco*. Or, plus l'ensemble des ingrédients substituant est grand, plus l'adaptation risque d'échouer, en raison d'une moins grande *ressemblance* entre les ingrédients.

Une analyse plus profonde des substitutions, recette par recette, permet de juger du succès ou de l'échec de l'adaptation et, le cas échéant, la cause de l'échec est identifiée. Table 3 donne pour chacune des recettes de TAAABLE$_0$, la liste des actions appliquées à l'ingrédient substitué. Si l'action n'est pas applicable à l'ingrédient substituant, dans notre cas le mascarpone, l'adaptation échoue ; ces échecs sont marqués par une croix dans la colonne *Échec*. Ainsi, dans les réponses retournées par TAAABLE$_0$, il y a 3 cas d'échecs nets, pour les recettes :

– *Pasta gratin with ...* dans laquelle il faudrait râper le mascarpone ;
– *Lasagne a la baroque* dans laquelle il faudrait couper le mascarpone en tranches ;
– *Southwest stew* dans laquelle il faudrait émietter le mascarpone.

Trois *demi*-échecs sont également à considérer dans les cas où TAAABLE$_0$ propose un choix dans l'ingrédient à substituer. Ainsi, l'adaptation de la recette *Pasta garden* peut être considérée comme un succès si on choisit de remplacer la ricotta par le mascarpone, mais comme un échec si on considère le remplacement de la mozzarella qui impliquerait que le mascarpone devrait être mis en lambeaux. Il en est de même pour la recette *No-fuss lasagna*, dont la substitution de ricotta conduit au succès, alors que la substitution de la mozzarella conduit à l'échec, et pour la l'adaptation de la recette *Eggplant parmigiana*, dont la substitution de ricotta conduit au succès, alors que la substitution de la mozzarella conduit à l'échec. Au total, 6 des 11 substitutions posent problème.

	Ingrédient substitué	Actions à subir	Échec
Eggplant parmigiana	Ricotta Mozzarella	ajouté coupé	 ×
Pasta gratin with ...	Mozzarella	rapé	×
Pasta garden pie	Mozzarella Ricotta	mis en lambeaux, puis parsemé remué	×
Tex mex lasagna	NonFatCottageCheese	mixé	
No-fuss lasagna	Mozzarella Ricotta	mis en lambeaux mixé	×
Lasagne a la baroque	Mozzarella	coupé en tranche, ajouté	×
3-step veggie pizza	CreamCheese	parsemé	
Southwest stew	QuesoFresco	émietté	×

TABLE 3 – Liste des actions appliquées aux ingrédients substitués, mentionnées dans les recettes avec tomate et mascarpone, retournées par TAAABLE$_0$. Une croix dans la colonne *Échec* indique que l'adaptation échoue car il n'est pas possible d'appliquer l'action à l'ingrédient substituant.

	Ingrédient substitué	Actions à subir	Échec
Pasta garden pie	Ricotta	remué	
No-fuss lasagna	Ricotta	mixé	
3-step veggie pizza	CreamCheese	parsemé	
Eggplant parmigiana	Ricotta	ajouté	

TABLE 4 – Liste des actions appliquées aux ingrédients substitués, mentionnées dans les recettes avec tomate et mascarpone, retournées par TAAABLE$_1$. Une croix dans la colonne *Échec* indique que l'adaptation échoue car il n'est pas possible d'appliquer l'action à l'ingrédient substituant. Pour TAAABLE$_1$, il n'y a aucun échec d'adaptation.

Table 4 donne pour chacune des recettes de TAAABLE$_1$, la liste des actions appliquées à l'ingrédient substitué, et mentionne si l'adaptation échoue ou non. L'ontologie raffinée améliore sensiblement le résultat final puisque toutes les adaptations de TAAABLE$_0$ qui échouaient ne sont plus proposées par TAAABLE$_1$: il n'y a aucun cas où l'adaptation échoue. Cependant, une des recettes pour laquelle l'adaptation réussissait dans TAAABLE$_0$ (*Tex-mex lasagna*) n'est plus proposée dans TAAABLE$_1$, en raison du nouvel éloignement de `NonFatCottageCheese` par rapport à `Mascarpone` dans l'ontologie de TAAABLE$_1$. Ceci est inhérent à la condition d'arrêt de l'algorithme de parcours de l'espace des généralisations de TAAABLE. Ce problème est résolu par la possibilité de demander, dans l'interface du système, de relancer la recherche des recettes similaires, au-delà de la généralisation courante, au risque, à nouveau, d'introduire des adaptations impossibles.

6 Conclusion

Nous avons montré, dans cet article, comment un raffinement de l'ontologie du domaine a un impact direct sur le processus de remémoration d'un système de RÀPC et améliore le résultat final, dans le cadre du système TAAABLE. Notre approche permet de remémorer plus progressivement les cas et, dans le cadre de TAAABLE, les cas d'adaptation problématiques sont éliminés.

Au delà des résultats de notre approche, illustrée précédemment sur un exemple concret, ce travail soulève plusieurs points qui nécessitent d'être étudiés et améliorés :

– le contrôle des propriétés : nous avons choisi, dans ce travail, de contrôler la liste des propriétés prises en compte lors du raffinement de l'ontologie, car le nombre d'actions culinaires est supérieur à 250. Choisir, *a priori*, des propriétés (actions) qui discriminent entre eux les ingrédients restreint certes les risques de regroupements indésirables. Cependant, comme les contextes d'utilisation d'un même ingrédient peuvent changer, il ne serait pas impossible, étant donné un ensemble d'actions A_1, qu'un ingrédient I soit plus proche d'un ingrédient I_1 et qu'étant donné un autre ensemble d'actions $A_2 \neq A_1$, I soit plus proche de I_2 (avec $I_1 \neq I_2$). Ainsi, dans un contexte, la ricotta pourrait être proche du mascarpone et dans un autre plus proche du *cottage cheese*.

– la représentation des cas : chaque cas (recette) est actuellement indexée par les ingrédients qu'elle nécessite, indépendamment de comment il est utilisé. Améliorer la représentation des cas, en les indexant par l'ingrédient et ses transformations est sans doute une autre façon de limiter le problème d'échec dans l'adaptation.
– l'amélioration de l'adaptation : un autre point envisagé dans l'amélioration de TAAABLE est de travailler sur les séquences d'actions et de garantir la faisabilité de la suite des actions après adaptation, par l'utilisation de pré-conditions fondées sur des connaissances du domaine : état de l'aliment (*solide*, *liquide*, *mou*, etc.), propriétés organoleptiques de l'aliment (*sucré*, *salé*, etc.), etc. Ces connaissances permettront de vérifier, après adaptation, que la recette est *cuisinable*.

Références

[1] F. Badra, R. Bendaoud, R. Bentebitel, P.-A. Champin, J. Cojan, A. Cordier, S. Després, S. Jean-Daubias, J. Lieber, T. Meilender, A. Mille, E. Nauer, A. Napoli, et Y. Toussaint. Taaable : Text Mining, Ontology Engineering, and Hierarchical Classification for Textual Case-Based Cooking. In *ECCBR Workshops, Workshop of the First Computer Cooking Contest*, pages 219–228, 2008.

[2] F. Badra, J. Cojan, A. Cordier, J. Lieber, T. Meilender, A. Mille, P. Molli, E. Nauer, A. Napoli, H. Skaf-Molli, et Y. Toussaint. Knowledge Acquisition and Discovery for the Textual Case-Based Cooking system WikiTaaable. In *Workshops of the 8th International Conference on Case-Based Reasoning (ICCBR-09)*, 2009.

[3] R. Bendaoud, A. Napoli, et Y. Toussaint. Formal Concept Analysis : A unified framework for building and refining ontologies. In Aldo Gangemi et Jérôme Euzenat, editors, *16th International Conference on Knowledge Engineering and Knowledge Management - EKAW 2008*, volume 5268, pages 156–171, Acitrezza, Catania Italie, 2008. Springer Berlin / Heidelberg.

[4] P. Cimiano, A. Hotho, et S. Staab. Learning concept hierarchies from text corpora using formal concept analysis. In *Journal of Artificial Intelligence Research (JAIR'05)*, volume Volume 24, pages 305–339. AAAI Press, 2005.

[5] V. Dufour-Lussier, J. Lieber, E. Nauer, et Y. Toussaint. Text adaptation using formal concept analysis. In *Case-Based Reasoning Research and Development (proceedings of ICCBR-2010, to be published)*, Lecture Notes in Artificial Intelligence. Springer, 2010.

[6] B. Ganter et R. Wille. *Formal Concept Analysis : Mathematical Foundations*. Springer, Berlin, 1999.

[7] J. Lieber. Le moteur de raisonnement à partir de cas de WIKITAAABLE. In *Actes du 17ème atelier raisonnement à partir de cas (RàPC-09)*, 2009.

[8] G. Stumme et A. Maedche. Fca-merge : Bottom-up merging of ontologies. In *17th International Joint Conferences on Artificial Intelligence (IJCAI'01)*, pages 225–234, San Francisco, CA, 2001. Morgan Kaufmann Publishers, Inc.

[9] L. Szathmary et A. Napoli. CORON : A Framework for Levelwise Itemset Mining Algorithms. *Supplementary Proceedings of The Third International Conference on Formal Concept Analysis (ICFCA '05), Lens, France*, pages 110–113, 2005.

Analyse formelle de concepts pour l'adaptation de cas textuels

Valmi Dufour-Lussier, Jean Lieber, Emmanuel Nauer, Yannick Toussaint
LORIA – UMR 7503 (CNRS, INRIA, Nancy-Université)
BP 239, 54506 Vandœuvre-lès-Nancy, France
{prénom.nom}@loria.fr

Résumé

Nous proposons une approche pour l'adaptation de cas textuels à l'aide de l'analyse formelle de concepts et de techniques empruntées au traitement automatique des langues naturelles. Les cas en question sont des recettes de cuisine, dans lesquelles nous cherchons à classer les ingrédients selon les techniques culinaires qui y sont appliquées. La nature complexe des anaphores dans les recettes rendent les techniques classiques de fouille de textes inefficaces, nous obligeant à adopter une approche plus forte, utilisant une analyse syntaxique et sémantique dynamique afin de construire une représentation formelle de chaque recette. Cette représentation peut avoir diverses utilités mais, dans cet article, nous montrons de quelle manière on peut y extraire des couples ingrédient–action et procéder à une analyse formelle de concepts pour sélectionner une séquence d'actions culinaires appropriées à utiliser dans l'adaptation d'un texte de recette où un ingrédient a été remplacé par un autre.

Mots clés : analyse formelle de concepts, fouille de textes, raisonnement à partir de cas textuels, traitement automatique des langues naturelles

1 Introduction

Adapter un cas remémoré pour résoudre un problème cible est une étape classique dans un système de raisonnement à partir de cas (RÀPC). L'adaptation d'un cas textuel peut consister simplement à remplacer toutes les occurrences d'un mot par un autre, mais nous pouvons imaginer faire mieux. Les participants au *Computer Cooking Contest*[1] *(CCC)* appliquent le RÀPC sur un ensemble de recettes (vues comme des cas). La solution à un problème de cuisine (adapter une recette à certaines contraintes) consiste jusqu'à ce jour à proposer des substitutions d'ingrédients, sans modifier le texte de la préparation de la recette.

Ce papier montre comment la fouille de textes par analyse formelle de concepts (AFC) est employée pour l'adaptation textuelle. Les *prototypes* de préparation d'un ingrédient sont extraits et utilisés pour adapter une recette. Adapter une recette en remplaçant un ingrédient α par un ingrédient β implique d'identifier les actions appliquées α et de les remplacer par des actions applicables à β.

Ce travail a été réalisé dans le cadre du projet TAAABLE [3, 7] ; il se focalise sur l'adaptation textuelle. TAAABLE est un système de RÀPC textuel dans le domaine de la cuisine qui a participé aux deux premiers CCC. Un moteur d'inférences exploitant une représentation des recettes sous forme de logique propositionnelle, un ensemble de substitutions potentielles d'ingrédients et une ontologie d'ingrédients permettent de proposer de nouvelles substitutions. Une fonction de coût permet de sélectionner la meilleure adaptation pour un problème donné.

Dans cet article, nous plaidons pour une représentation formelle plus approfondies des recettes et montrons comment elle peut être construite à partir de techniques de traitement automatique de la langue (TAL) et comment l'AFC permet d'exploiter cette nouvelle représentation pour améliorer l'adaptation. Partant du fait que TAAABLE est capable de proposer une recette de sa base de cas et les substitutions consistant à remplacer un ingrédient par un autre, notre système est capable de trouver une séquence d'actions à réaliser sur l'ingrédient substituant pour mieux l'intégrer à la recette. Le texte de la recette est également modifié en conséquence.

[1] http://vm.liris.cnrs.fr/ccc2010

Easy berry pancakes

Ingredients	Preparation
•Six eggs	Beat the eggs. Add the flour and mix with a
•2 cups of flour	fork. Whisk in the milk.Pour batter in warm
•2 cups of milk	pan, one ladleful at a time. Add some fruits,
•½ cup of blueberries	then cook for one minute. Flip and cook one
•½ cup of raspberries	minute more.

FIG. 1: Exemple d'un texte de recette.

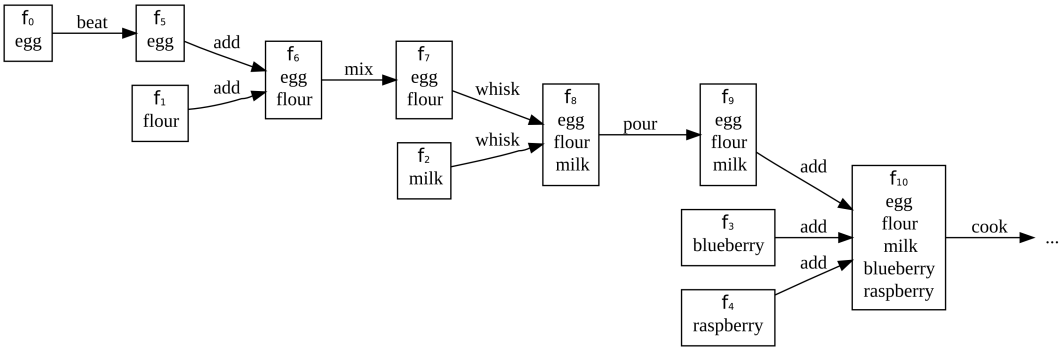


FIG. 2: Représentation sous forme d'arbre de la recette de la figure 1.

Bien que la sélection des textes utilisant l'AFC et leur réutilisation dans le processus d'adaptation d'un système de RÀPC textuel est, à notre connaissance, une nouveauté, cette approche correspond à l'idée d'une réutilisation maximale des textes afin de produire des solutions textuelles aux problèmes, comme cela est défendu dans [14, 13]. Utiliser l'AFC pour la recherche d'information ou le RÀPC n'est pas une idée tout à fait neuve (voir [6, 16] pour la recherche d'information et [9, 8] pour le RÀPC).

Dans la section 2, nous présentons le formalisme dans lequel nous voulons représenter les recettes et le processus pour traduire les textes dans cette représentation. Puis, dans la section 3, nous montrons comment l'AFC est utilisée pour l'adaptation des recettes, et nous détaillons les algorithmes que nous avons développés ainsi que la stratégie utilisée pour générer deux nouveaux textes. Finalement, nous discutons des résultats et présentons des perspectives de recherches dans 4 et 5.

2 Analyse linguistique des recettes

L'idée principale guidant ce travail est que des « usages courants » de chaque ingrédient, que nous appelons *prototypes*, peuvent être extraits de la base de cas et employés pour l'adaptation de textes. Si par exemple on veut remplacer la *courgette* par de l'*aubergine* dans une recette, il serait plus pertinent de préparer l'aubergine de la même manière que dans une recette d'aubergine connue que de lui appliquer aveuglément les mêmes actions culinaires qu'à la courgette. Un prototype s'entend comme une séquence d'actions appliquées à un ingrédient. Pour le connaître, nous avons besoin d'une représentation formelle du texte de recette. Le traitement linguistique qui permet de créer la représentation d'un texte est appliqué à la fois sur la recette source (celle qui doit être adaptée) et sur toutes les recettes de la base de cas. Ces représentations sont ensuite transmises à un algorithme de fouille de données pour extraire les prototypes. Le processus complet sera illustré à l'aide de la recette présentée à la figure 1, avec pour résultat la représentation de la figure 2.

2.1 Représenter les recettes sous forme d'arbre

Les recettes sont des textes procéduraux composés d'une séquence d'actions à travers laquelle les ingrédients sont progressivement combinés pour obtenir un seul produit final, le plat. La plupart de ces actions

culinaires sont exprimées par des verbes. Chaque action prend comme argument un ou plusieurs « aliments » pour produire, en sortie, un nouvel aliment.

Pour adapter une recette, il peut être utile de la diviser en petites « parties » de telle sorte que certaines de celles-ci puissent être remplacées par de nouvelles parties. En conséquence, la représentation formelle d'une recette doit laisser apparaître nettement les différentes étapes d'une recette et les « régularités » entre des ensembles de recettes. Si on voit les actions culinaires comme des fonctions appliquées sur des aliments, une modélisation des recettes sous forme d'arbre apparaît comme naturelle. On peut aussi voir cela comme un parti pris radical dans le cadre des théorie de Asher sur la sémantique dynamique [1], considérant chaque verbe simultanément comme un verbe de destruction et de construction (d'aliments). Certaines situations rendent en fait une représentation arborée inadéquate, comme l'action qui consiste à séparer un œuf en blanc et jaune. Nous n'avons pas tenu compte de ce genre de cas, mais croyons néanmoins que notre approche pourrait facilement être généralisée pour une forme de représentation utilisant des graphes dirigés acycliques.

Dans la représentation arborescente d'une recette de cuisine, les feuilles correspondent donc aux ingrédients initiaux, la racine au plat cuisiné et chaque autre nœud aux états subséquents des différents aliments. Chaque nœud est étiqueté par un label unique ℓ, et on définit une fonction $\mathcal{I}(\ell)$ donnant l'ensemble des ingrédients dont l'aliment ℓ est composé. Par exemple, dans la figure 2, $\mathcal{I}(\mathrm{f}_6) = \{\mathrm{egg}, \mathrm{flour}\}$.

C'est à cet structure en arbre que le processus de fouille de données est appliqué. Cela dit, l'arbre est aussi nécessaire à la résolution de certains problèmes linguistiques posés par les textes de recette. L'arbre est construit par un processus itératif : les actions trouvées dans le texte sont traitées dans l'ordre, l'une après l'autre, chacune ajoutant un nœud à l'arbre en utilisant l'information qui y est déjà présente.

2.2 La spécificité des textes de recette

Bien que les textes de recette présentent l'avantage d'une grande régularité, ils présentent aussi des difficultés spécifiques inhérentes à leur nature procédurale :

1. Ils font un grand usage de structures de phrase comme des propositions impératives, rares dans d'autres types de textes, ce qui rend les outils entraînés sur des corpus génériques inefficaces.
2. Ils utilisent massivement un phénomène linguistique peu étudié connu sous le nom d'*anaphore évolutive*, où un mot est employé pour référer à un objet qui existe à un moment donné, mais pas (ou plus) à un autre, nécessitant une stratégie spéciale pour découvrir ce à quoi ce mot réfère à un moment donné, par exemple, « mélanger la farine, les œufs et le lait ; verser *la pâte* » ;
3. Pour éviter de lourdes répétitions, particulièrement en anglais, ils omettent généralement les arguments syntaxiques des verbes qui semblent évidents, requérant une stratégie pour déterminer si des arguments manquent et, le cas échéant, à quoi l'argument manquant correspond, par exemple « cook potatoes ; when done, add milk [implicitement : *to potatoes*] » (c'est l'*ellipse*, un type d'anaphore grammaticale).

2.3 La chaîne de traitement

Les premières étapes du traitement linguistique peuvent être effectuées à l'aide de méthodes bien connues du traitement automatique des langues naturelles. Bien que nous ne puissions pas utiliser de corpus annoté autant que normal, nous avons pu obtenir un petit corpus d'une centaine de recettes annotées avec la partie du discours (verbe, nom, adjectif…) de chaque mot, ce qui a suffi pour l'entraînement d'un étiqueteur transformationnel sensible au contexte [5] avec une précision $p > 0,90$, bien en-deçà de l'état de l'art, mais néanmoins acceptable pour un prototype. Les autres étapes préliminaires (segmentation en mots et en propositions, analyse syntaxique superficielle) ont été mises en œuvre avec des expressions régulières définies manuellement.[2]

À cette étape, nous savons quels mots sont des verbes (donc des actions) et somment capables d'identifier leurs arguments syntaxiques, ce qui est suffisant pour initier la construction de la représentation formelle arborescente d'une recette.

[2] Par exemple, l'expression régulière reconnaissant un syntagme nominal est "N′((Virgule N′)*(Virgule|Conjonction)$^{\{1,2\}}$N′)$^?$", où "N′" correspond à "Prédéterminant$^?$ Déterminant$^?$ Adjectif* Nom$^+$".

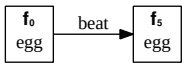


FIG. 3: Œufs battus.

2.4 Du texte à l'arbre

Les problèmes décrits dans les paragraphes 2.2(2) et 2.2(3) ne peuvent pas être résolus à l'échelle de la phrase. Par contre, si on les néglige, la représentation obtenue sera un ensemble d'arbres impossibles à connecter, chacun représentant une partie de la recette. Une étape de post-traitement spécifique a donc été conçue.

On part de l'idée que, initialement, tous les ingrédients sont disponibles pour être sélectionnés comme argument par la toute première action. De la même manière, à toute étape de la recette, un certain ensemble d'aliments sont disponibles et l'action suivante en sélectionne un ou plusieurs afin d'en créer un nouveau.

Nous nommerons l'ensemble des aliments disponibles pour subir une action culinaire le *domaine*. Le domaine initial $\mathcal{D}_0$ est un ensemble renfermant un aliment correspondant à chacun des ingrédients de la liste des ingrédients. Le domaine est modifié après l'accomplissement de chaque action : les ingrédients utilisés, s'ils ont été transformés, ne font plus partie du nouveau domaine, en revanche, le nouvel aliment produit est ajouté au domaine. Ainsi, le domaine après t actions, noté $\mathcal{D}_t$, est employé pour trouver les arguments de la $t+1^{\text{ème}}$ action.

Dans le cas de l'exemple de la figure 1, le domaine initial est le suivant :

$$\begin{aligned}\mathcal{D}_0 =&\{\mathrm{f}_0, \mathrm{f}_1, \mathrm{f}_2, \mathrm{f}_3, \mathrm{f}_4\},\\ &\mathcal{I}(\mathrm{f}_0) = \{\text{egg}\},\\ &\mathcal{I}(\mathrm{f}_1) = \{\text{flour}\}, \text{etc.}\end{aligned}$$

Pour une action avec un seul argument, un arc étiqueté par l'action est créé entre cet argument et un nouveau nœud représentant un nouvel aliment. Le domaine est alors modifié, de telle sorte que pour un verbe comme « beat X », où X est l'aliment auquel est appliqué l'action, tel que déterminé par l'algorithme que nous présentons, on obtient :

$$\begin{aligned}\mathcal{D}_t =&(\mathcal{D}_{t-1}\backslash\{X\}) \cup \{\ell\},\\ &\mathcal{I}(\ell) = \mathcal{I}(X)\ ,\end{aligned} \tag{1}$$

où ℓ est un nouvel aliment. Par exemple, la première phrase de la recette–exemple est « beat the eggs », ce qui a pour effet de créer un nouvel aliment « egg » (voir figure 3) :

$$\begin{aligned}\mathcal{D}_1 =&(\mathcal{D}_0\backslash\{\mathrm{f}_0\}) \cup \{\mathrm{f}_5\},\\ &\mathcal{I}(\mathrm{f}_5) = \mathcal{I}(\mathrm{f}_0)\ .\end{aligned}$$

Quant aux actions à plusieurs arguments, elles peuvent avoir une sémantique conduisant au rattachement à un même nœud de deux arbres, comme « add X to Y »,

$$\begin{aligned}\mathcal{D}_t =&(\mathcal{D}_{t-1}\backslash(\{X\} \cup \{Y\})) \cup \{\ell\},\\ &\mathcal{I}(\ell) = \mathcal{I}(X) \cup \mathcal{I}(Y)\ ,\end{aligned} \tag{2}$$

ou, au contraire, une sémantique conduisant à la création à partir d'un nœud de deux arbres différents, comme « remove χ from X », où χ est un ingrédient (pas un aliment) :

$$\begin{aligned}\mathcal{D}_t =&(\mathcal{D}_{t-1}\backslash\{X\}) \cup \{\ell\},\\ &\mathcal{I}(\ell) = \mathcal{I}(X)\backslash\{\chi\}\ ,\end{aligned} \tag{3}$$

où ℓ est un nouvel aliment. Par exemple, la phrase suivante de la recette–exemple est « add the flour », une action « du premier type » (voir figure 4) :

$$\begin{aligned}\mathcal{D}_2 =&(\mathcal{D}_1\backslash(\{\mathrm{f}_1\} \cup \{\mathrm{f}_5\})) \cup \{\mathrm{f}_6\},\\ &\mathcal{I}(\mathrm{f}_6) = \mathcal{I}(\mathrm{f}_1) \cup \mathcal{I}(\mathrm{f}_5)\ .\end{aligned}$$

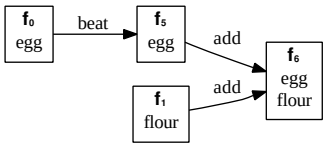


FIG. 4: Ellipse résolue : « Add flour [to eggs] ».

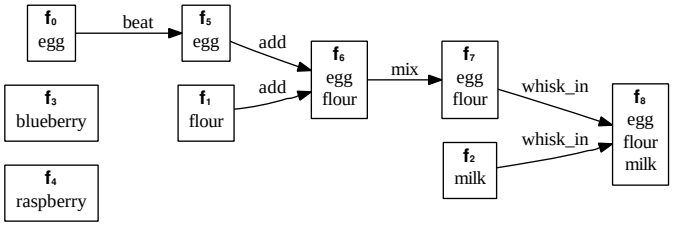


FIG. 5: Les aliments du domaine étant f_3, f_4 et f_8, la formule (4) avec $\mathcal{T} = \{\text{egg}, \text{flour}\}$ nous indique que le mot « batter » ne peut faire référence qu'à f_8.

Un dictionnaire de sous-catégorisation des actions a été construit manuellement pour décrire les 128 verbes de notre corpus. Il nous indique le ou les types d'arguments requis, nous permettant de définir une stratégie simple mais efficace pour identifier les ellipses. Si un argument semble manquer, le dernier nœud ajouté à l'arbre est systématiquement considéré comme jouant le rôle de cet argument. C'est ainsi que l'on infère que, dans la recette de la figure 1, c'est aux œufs que la farine est ajoutée pour produire le sous-arbre présentée en figure 4.

Le domaine représentant l'ensemble des aliments disponibles à une étape est utile pour la résolution d'anaphores plus complexes. Le traitement des deux cas les plus couramment rencontrés dans les recettes est maintenant détaillé.

2.4.1 Références existentielles

Une expression comme « beef mixture » réfère à un aliment contenant au moins un ingrédient donné, en l'occurrence du bœuf, possiblement mélangé à d'autres. Il est donc nécessaire de chercher dans le domaine un aliment comprenant au moins cet ingrédient. Nous définissons donc un ensemble $\mathcal{T}$ d'« ingrédients cibles » et une fonction simple permettant de retrouver l'aliment auquel l'expression anaphorique se réfère :

$$x \in \mathcal{D} : \exists i.i \in \mathcal{I}(x) \wedge i \in \mathcal{T} \ . \tag{4}$$

Dans le cas de « beef mixture », cela est trivial : $\mathcal{T} = \{\text{beef}\}$. Mais certains cas sont plus délicats : nous avons calculé à partir d'une exploration du corpus que l'aliment auquel on se réfère par le mot « batter » (pâte à crêpe) contient, dans plus de 99% des cas, des œufs ou de la farine. On définit donc $\mathcal{T} = \{\text{egg}, \text{flour}\}$. Ainsi, l'instruction « pour batter » de la recette–exemple peut être traitée. Avant « exécution » de l'action « pour », l'arbre résultant du début de la recette est donné en figure 5. En observant la liste des ingrédients pour chacun des aliments du domaine, il ressort clairement que f_8 est l'aliment référencé par l'expression « batter ».

2.4.2 Références universelles

D'autres expressions, de nature hyperonymique, réfèrent manifestement à un ensemble d'ingrédients appartenant à une certaine classe. C'est le cas, par exemple, du mot « fruits » lorsque la liste d'ingrédients ne contient en fait rien qui s'appelle « fruit », mais contient des myrtilles et des framboises. Puisqu'une ontologie d'ingrédients est déjà présente dans le système TAAABLE, nous l'utilisons pour identifier l'ensemble d'aliments désignés par le nom de leur classe :

$$\{x \in \mathcal{D} : \forall i.i \in \mathcal{I}(x) \rightarrow i \sqsubseteq \text{Fruit}\} \ ,$$

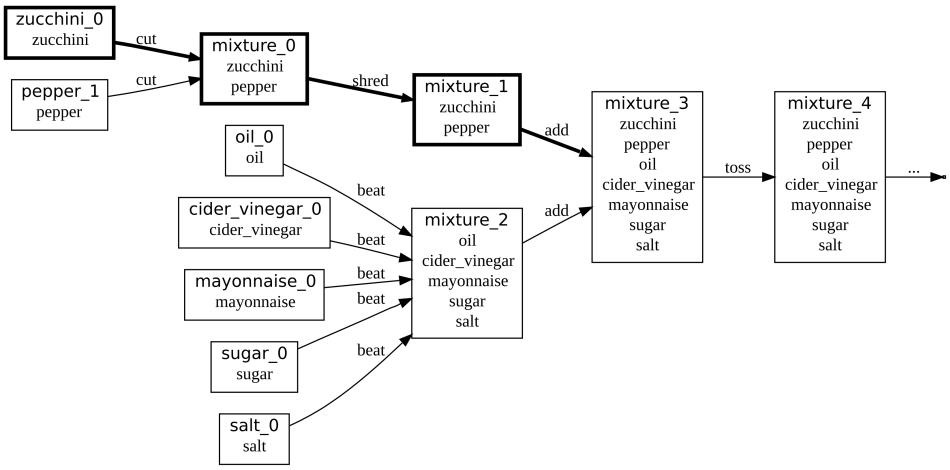


FIG. 6: Le sous-arbre pertinent à l'ingrédient « zucchini » dans la recette donnée en figure 9 identifié en gras.

où « $i \sqsubseteq$ Fruit » signifie que l'ingrédient i appartient a la classe « Fruit » de l'ontologie. Donc l'instruction « add some fruits » réfère à $\{\mathrm{f}_3, \mathrm{f}_4\}$.

3 Analyse formelle de concepts pour l'adaptation de recettes

Une fois les recettes formalisées sous forme d'arbres, pour chaque ingrédient et pour chaque recette, il existe un chemin (une séquence d'actions) entre cet ingrédient et l'état final de la recette qui représente le plat.

Le processus d'adaptation peut alors être redéfini. Adapter une recette par substitution d'un ingrédient α par un ingrédient β consiste à remplacer le sous-arbre de l'ingrédient α de la recette remémorée par un sous-arbre – le plus approprié – de l'ingrédient substituant β extrait d'une autre recette.

Sur la base d'un exemple réel extrait du corpus du *Computer Cooking Contest*, nous détaillons maintenant comment sont définis le sous-arbre à remplacer et le sous-arbre remplaçant. Supposons que l'on veuille une salade froide à l'aubergine et que le système remémore une recette de salade froide à la courgette et propose de remplacer la courgette par l'aubergine. Le système doit alors rechercher dans les recettes disponibles, l'instance d'aubergine dont le mode de préparation est le plus proche de celui de la courgette dans la salade à la courgette. L'analyse formelle de concepts construit une conceptualisation des différentes manières de préparer les aubergines de telle sorte que le concept le plus proche de la préparation de la courgette peut être facilement identifié.

3.1 Extraire le sous-arbre pertinent

Nous devons maintenant extraire le sous-arbre pertinent à l'ingrédient qui nous intéresse de la représentation (ceci sera effectué à la fois pour l'ingrédient substitué et l'ingrédient de substitution). Trois types d'actions sont considérés : celles appliquées à un ingrédient seul, celles appliquées à plusieurs ingrédients en parallèle et celles appliquées à plusieurs ingrédients considérés comme un tout. La distinction entre le deuxième et le troisième type est importante : alors que des pommes et des poires pelées demeurent distinctement des pommes *et* des poires, des pommes et des poires cuites ensemble constituent une mixture qui n'a plus grand chose en commun avec les ingrédients d'origine, et à laquelle un ensemble d'actions éventuellement différent peut être appliqué. De plus, si toute l'information disponible sur un ingrédient est prise en compte, l'espace des actions devient trop large et il n'a plus assez de concentration des données pour permettre un apprentissage efficace.

Puisqu'aucun indice linguistique n'est disponible pour classer les actions entre les deuxième et troisième types, nous utilisons simplement une liste des actions possibles indiquant leur type le plus probable. Si on considère maintenant une recette réelle prise dans la base de cas, représentée à la figure 6, « toss » est une action de la troisième catégorie, donc les actions appliquées à la courgette sont considérées comme « intéressantes » jusqu'à « add », donc le prototype de courgette employé dans cette recette est « zucchini $\xrightarrow{\text{cut}} \cdots \xrightarrow{\text{shred}} \cdots \xrightarrow{\text{add}}$ ».

	add	arrange	bake	beat	blend	boil	break	...
aubergine_509	x							
aubergine_667	x		x		x			
aubergine_981		x						
aubergine_1030			x					
aubergine_1387			x					
...								
zucchini	x							

TAB. 1: Le contexte formel pour l'espace de la requête.

Lorsque la substitution sera effectuée ultérieurement, le prototype de courgette sera donc détaché au niveau de la « mixture_3 » et le prototype d'aubergine choisi à la section 3.2 sera rattaché à ce même point.

3.2 Analyse formelle de concepts

L'AFC [10] prend en entrée une table binaire, comme celle présentée en table 1, décrivant une relation binaire entre des objets (« aubergine_509 », « aubergine_667 » ...) et des attributs (« add », « arrange » ...). Cette table est également appelée *contexte formel*, et se définit par $\mathbb{K} = \langle \mathcal{O}, \mathcal{A}, I \rangle$, avec $\mathcal{O}$ un ensemble d'objets, $\mathcal{A}$ un ensemble d'attributs, et $I \subseteq \mathcal{O} \times \mathcal{A}$ la relation binaire telle que $\langle o, a \rangle \in I$ signifie que l'objet o possède l'attribut a. Une connexion de Galois est définie par les fonctions suivantes : $f : 2^{\mathcal{O}} \rightarrow 2^{\mathcal{A}}$ et $g : 2^{\mathcal{A}} \rightarrow 2^{\mathcal{O}}$, par lesquelles un ensemble d'objets est mis en relation avec l'ensemble des attributs qu'ils ont en commun, et réciproquement. Les concepts formels sont des couples de $O \times A$, fermés par la connexion de Galois, qui permettent de construire un treillis de concept, noté $\mathfrak{L}(\mathbb{K})$. Chaque nœud du treillis, comme celui présenté en figure 7 représente un concept formel, qui est un couple $\langle O, A \rangle$, avec O, l'ensemble des objets possédant tous les attributs de A, et A, l'ensemble des attributs possédés par tous les objets de O. O et A sont respectivement appelés *extension* et *intension* du concept formel. Le treillis est partiellement ordonné par l'inclusion des extensions ; si l'extension d'un concept C est incluse dans l'extension d'un concept D (avec $C \neq D$), alors C est dessiné en-dessous de D, un chemin relie C à D, et on dit que D subsume C.

Le mode de préparation d'un ingrédient étant caractérisé par les actions culinaires qui lui sont appliquées, le contexte formel qui est généré utilise des actions culinaires appliquées aux aubergines comme attributs, chaque objet étant une recette employant de l'aubergine. Dans notre cas (issu d'une base avec peu de recettes pour permettre la lisibilité du treillis), le contexte formel est composé de 15 objets et 55 attributs ; un extrait est donné en Table 1, avec l'ajout d'un objet correspondant à la recette de zucchini et des actions culinaires sollicitées. Le treillis résultant de ce contexte est donné en Fig. 7.

Le concept formel numéroté 1, appelé « Top », est le maximum du treillis. Son extension est l'ensemble de tous les objets et son intension est l'ensemble de tous les attributs possédés par tous les objets (dans ce cas, il s'agit de l'ensemble vide). Le concept 58, appelé « Bottom », est le minimum du treillis. Son intension est l'ensemble de tous les attributs et son extension est l'ensemble des objets qui possèdent tous les attributs (là encore, il s'agit de l'ensemble vide). Du « Top » au « Bottom », les concepts sont progressivement plus « spécifiques », signifiant que leur extension contient moins d'objets, mais qu'ils partagent plus d'attributs. Le concept 29 est le concept le plus spécifique contenant `zucchini` dans son extension : son extension contient tous les attributs de cet objet. Le concept 16, qui subsume directement le concept 2, est le concept le plus spécifique ayant `zucchini` ainsi que d'autres objets dans son extension (dans cet exemple, il y a seulement un concept qui subsume immédiatement le concept 29, mais il pourrait y en avoir plusieurs). D'une certaine façon, ces objets sont ceux qui sont les plus proches de l'objet `zucchini`, puisqu'ils ont le plus d'attributs en commun avec lui.

3.3 Interrogation du treillis

Comme le propose Carpineto [6], les différents prototypes d'adaptation de la recette sont ordonnés en exploitant le treillis. Dans l'exemple précédent de l'aubergine et de la courgette, la première étape cherche un

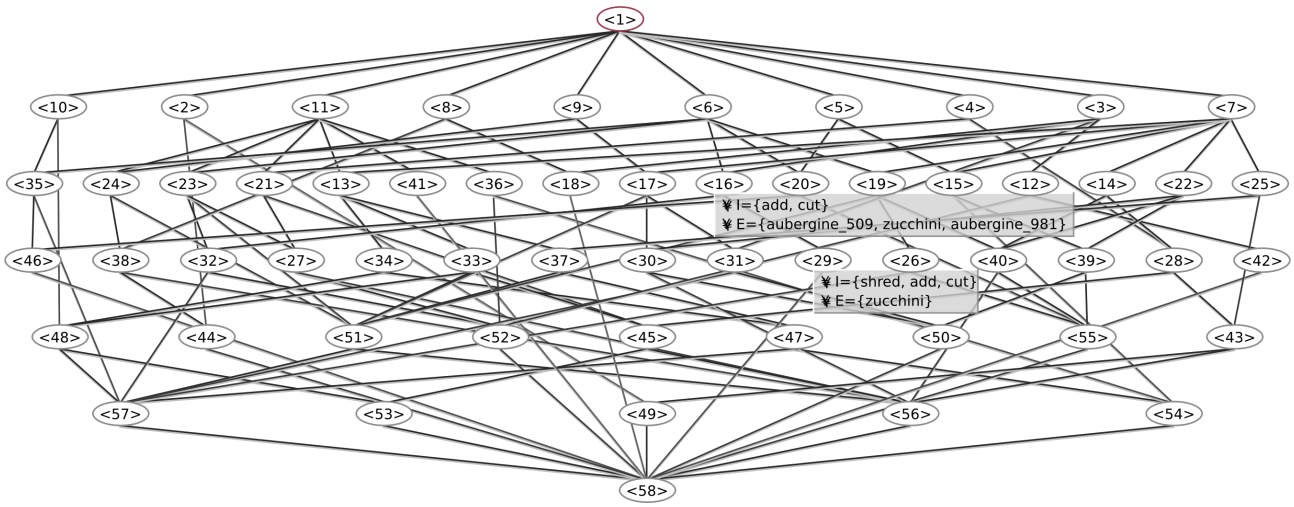


FIG. 7: Le treillis de concept issu du contexte donné en Table 1.

prototype d'aubergine compatible avec la préparation de la courgette, éventuellement, en incluant des étapes supplémentaires spécifiques à l'aubergine et excluant certaines étapes spécifiques à la courgette. Le mode de préparation de la courgette est alors vu comme une requête dans l'espace des modes de préparation des aubergines.

Nous recherchons les concepts les plus spécifiques dans le treillis pour lesquels la requête est incluse dans leur intension propre. Le candidat pour la substitution est alors choisi parmi les élément de l'extension de ce concept ou l'extension du concept le subsumant (à l'exclusion, bien sûr, de l'élément requête lui-même). Ainsi la recette qui minimise la différence entre la préparation de la courgette et celle de l'aubergine est la suivante :

$$\underset{\mathrm{a}}{\operatorname{argmin}} \left| \Big(\mathrm{Int}(\mathcal{C}_\mathrm{z}) \backslash \mathrm{Int}(\mathcal{C}_\mathrm{a}) \Big) \cup \Big(\mathrm{Int}(\mathcal{C}_\mathrm{a}) \backslash \mathrm{Int}(\mathcal{C}_\mathrm{z}) \Big) \right| \ , \tag{5}$$

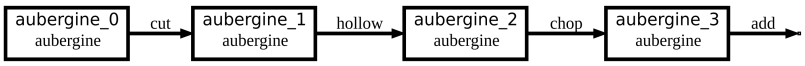
où $\mathcal{C}_\mathrm{a}$ et $\mathcal{C}_\mathrm{z}$ sont les concepts les plus spécifiques de a et z, et $\mathrm{Int}(\mathcal{C})$ est l'intension de $\mathcal{C}$. Dans notre exemple, on a 4 pour a = aubergine_509 et 3 pour a = aubergine_981. La formule (5) sélectionne donc aubergine_981.

3.4 L'algorithme d'adaptation

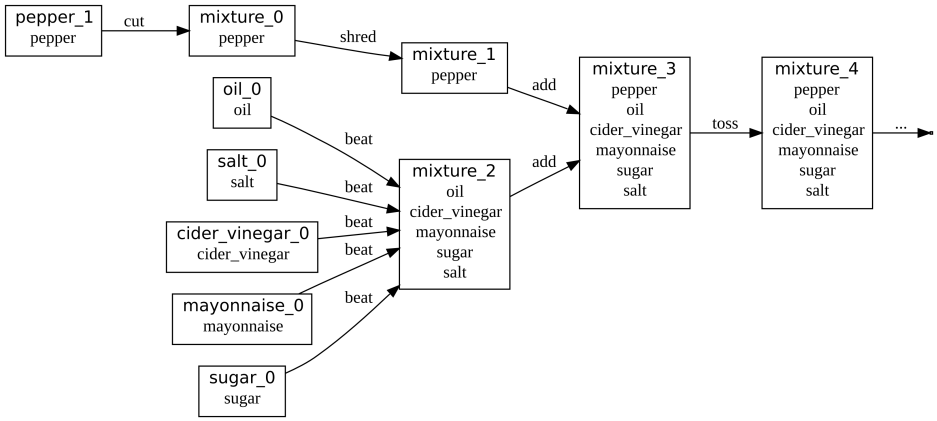
Tout ce qui reste à faire afin de finir l'adaptation de l'arbre est d'obtenir les sous-arbres de la courgette et de l'aubergine, en utilisant les techniques décrites dans la section 2.4, d'enlever le premier sous-arbre (en gardant les parties requises pour le processus sur d'autres ingrédients) et d'ajouter le deuxième.

On pourrait envisager d'utiliser des techniques d'adaptation en planification à partir de cas sur des plans ordonnés partiellement, tels que [20, 12, 18], afin d'intégrer de nouvelles étapes de préparation dans le reste de la recette. Cependant, ne pas conserver la structure de la recette le plus intacte possible empêcherait de réutiliser le texte tel qu'il est et obligerait à utiliser des techniques de génération de langage naturel, ce qui conduirait à des résultats de qualité textuelle médiocre [11]. De plus, la planification suppose des connaissances du domaine avancées, en particulier, les pré-conditions et post-conditions des actions doivent être connues. Par exemple, certains ingrédients, quand ils sont chauffés, créent une mousse qui doit être enlevée. En l'absence de telles connaissances, notre méthode de « greffage », nous semble une approche plus prudente. Une stratégie plus simple, qui consiste à connecter le nouveau sous-arbre à un seul point de l'arbre donne des résultats satisfaisants (ainsi que l'illustre la figure 8) et permet une adaptation textuelle plus simple.

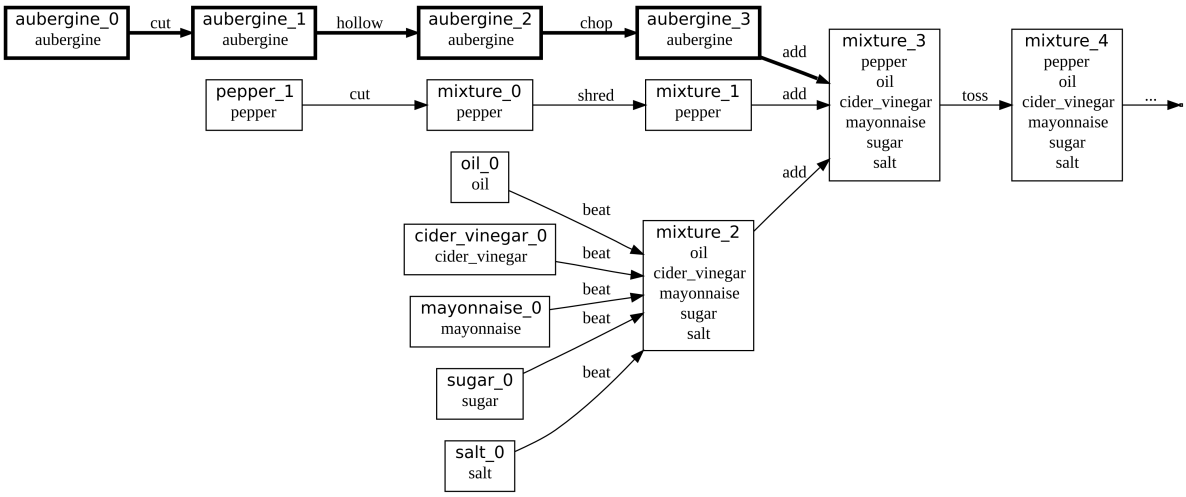
Nous proposons un algorithme pour l'adaptation textuelle qui est très simple et se déroule parallèlement au processus d'adaptation de l'arbre. Il copie chaque phrase de la recette sélectionnée autant que possible, ce qui donne au texte résultant un aspect plus « naturel » que s'il avait été généré automatiquement. Les actions relatives aux courgettes sont enlevées : si une phrase est relative uniquement à la courgette, elle est supprimée, sinon, seul le mot « courgette » est supprimé (et son déterminant, un mot de liaison tel que « et » ou une virgule). Puis, le texte relatif à la préparation de l'aubergine dans la recette sélectionnée est traitée de la même façon pour



(a) Le sous-arbre `aubergine_981` à ajouter.



(b) L'arbre de la figure 6 avec le sous-arbre des courgettes élagué.



(c) L'arbre final, en fin d'adaptation.

FIG. 8: Le processus de substitution de sous-arbre.

Cut the aubergines in half lengthwise. With a spoon, hollow out the center of each half to make a boat like shell about 1/4 inch thick. Finely chop the aubergine pulp and set it aside. Cut ~~zucchini and~~ red pepper into matchstick thin strips or shred in food processor ; add the chopped aubergine pulp. set aside. In large bowl, with wire whisk or fork, beat salad oil, vinegar, mayonnaise, sugar, salt and pepper until mixed. Add vegetables ; gently toss to mix well. [...]

FIG. 9: Le texte original de la recette avec les parties à enlever barrées et les parties à ajouter (copiées de la recette remémorée) soulignées.

enlever les références à tous les ingrédients en dehors des aubergines. Ce texte sur les aubergines est inséré au début du texte original, à part la clause contenant le dernier verbe du prototype, celui qui concerne le mélange avec les autres ingrédients, qui est insérée à l'endroit où tous ces autres ingrédients sont complètement traités. Ce processus est illustré à la figure 9. Il ne reste plus que quelques ajustements grammaticaux (p. ex. l'accord verbal) et typographiques.

4 Discussion et travaux similaires

Le processus d'adaptation donne de très bons résultats pour peu que les données extraites des textes soient de bonnes qualités. Le traitement linguistique des recettes est donc le maillon faible. La représentation arborescente d'une recette est de bonne qualité si les textes ne contiennent pas de faute d'orthographe, pas de structure complexes comme la négation (« do not peel the potatoes before boiling them ») ou l'élaboration (« boil the potatoes – but peel them first »). Ces phénomènes peuvent être pris en compte, avec plus ou moins de succès, mais ils compliquent nettement la mise en œuvre.

La chute de performance la plus significative est liée à l'étiquetage morpho-syntaxique (module qui détermine pour chaque mot s'il s'agit d'un verbe, d'un nom, etc.) car les recettes sont des textes particuliers pour lesquels nous ne disposons pas actuellement de corpus d'entraînement. Constituer un tel corpus est coûteux car un annotateur humain étiquette environ 3000 mots à l'heure [15] alors qu'il faut annoter quelques centaines de milliers de mots pour un résultat satisfaisant [4]. Une autre difficulté vient de la résolution des références. Ne pas relier un ingrédient à une action peut entraîner des erreurs en cascade fait que potentiellement, l'ingrédient ne sera pas associé à plusieurs des actions qui le concernent.

Environ 10% des représentations de recettes sont strictement sous la forme d'un arbre. On peut donc considérer ces situations comme une analyse quasi-parfaite et donc une représentation bien adaptée à notre processus d'adaptation. Pour les autres situations, la qualité est difficile à évaluer : l'analyse est généralement correcte pour les premières actions de la recette mais peut se dégrader très vite par la suite.

Évaluer un système de RÀPC est généralement très difficile, mais peut être fait à travers des comparaisons avec d'autres systèmes, et c'est une des motivations pour l'atelier CCC. Pour cette raison, l'algorithme d'adaptation décrit dans cet article est en cours d'implémentation dans le système TAAABLE et devrait donc être évalué par l'édition 2010 de ce concours.

L'AFC a déjà été utilisée en RÀPC, par exemple dans [8] et dans [2]. Dans [8], le contexte formel est une représentation de la base de cas (les objets sont les cas et leurs propriétés sont des propriétés binaires de ces cas). Le treillis de Galois obtenu par AFC structure la base de cas et le processus de remémoration consiste en une classification hiérarchique dans ce treillis. De plus, le treillis est utilisé pour assister le processus d'aide à la formulation de la requête (qu'on peut considérer comme une étape d'élaboration [17]). En revanche, dans notre travail, l'AFC est utilisée pour l'adaptation : le contexte formel concerne le vocabulaire de description des cas (les ingrédients et les actions effectuées sur les ingrédients). L'objectif de [2] est l'obtention de cas structurés à partir de textes, en utilisant l'AFC. Les objets du contexte formel sont des textes et ses propriétés sont des termes pertinents de ces textes. À la différence de notre application, dans les textes utilisés, chaque document peut être considéré comme un cas, alors que dans notre application, étant donné qu'on y trouve toujours plusieurs ingrédients, tous les mots-clés découverts dans un même texte ne sont pas pertinents pour une tâche d'adaptation donnée.

Beaucoup de cas, textuels ou non, ont une structure similaire à celles des recettes : les instructions d'assemblage, les manuels d'utilisateur et les instructions pour préparations pharmaceutiques sont des exemples de cas « procéduraux » pour lesquels l'approche décrite dans cet article est susceptible de s'appliquer.

5 Conclusion et travaux futurs

Nous proposons une solution pour adapter des textes dans un système de raisonnement à partir de cas. Nous utilisons des outils de traitement de la langue pour construire une représentation riche et formelle des textes puis des outils de fouille de données – l'analyse formelle de concepts – pour retrouver des textes dont des parties peuvent être réutilisées.

La représentation formelle des recettes peut être enrichie pour améliorer le système de raisonnement à partir de cas. Ainsi, il serait possible d'associer à certains nœuds de l'arbre de la recette des informations sur la texture, le volume (ou le poids), ou encore sur la saveur de l'objet culinaire associé à ce nœud. Ces informations peuvent facilement être intégrées à la structure de treillis pour être prises en compte dans la réponse.

Selon [19], le cas source retenu doit minimiser le coût de l'adaptation. Actuellement, ce coût est calculé en lien avec le parcours de la hiérarchie d'ingrédients. Par la suite, le coût pourrait prendre en compte l'adaptation de la préparation telle que nous la proposons. Si α et β sont deux ingrédients, plus il y a de prototypes communs dans le treillis, plus le coût de l'adaptation pourrait être bas.

Références

[1] N. Asher. Events, facts, propositions, and evolutive anaphora. In J. Higginbotham, F. Pianesi, et A.C. Varzi, editors, *Speaking of events*, pages 123–150. OUP, Oxford, 2000.

[2] S. Asiimwe, S. Craw, N. Wiratunga, et B. Taylor. Automatically Acquiring Structured Case Representations : The SMART Way. In *Applications and Innovations in Intelligent Systems XV*. Springer, 2007.

[3] F. Badra, R. Bendaoud, R. Bentebitel, P.A. Champin, J. Cojan, A. Cordier, S. Després, S. Jean-Daubias, J. Lieber, T. Meilender, A. Mille, E. Nauer, A. Napoli, et Y. Toussaint. TAAABLE : Text Mining, Ontology Engineering, and Hierarchical Classification for Textual Case-Based Cooking. In *ECCBR Workshops, Workshop of the First Computer Cooking Contest*, pages 219–228, Heidelberg, 2008. Springer.

[4] M. Banko et E. Brill. Mitigating the paucity-of-data problem : Exploring the effect of training corpus size on classifier. In *Proceedings of the first international conference on Human language technology research*, Morristown, 2001. Association for Computational Linguistics.

[5] E. Brill. *Transformation-Based Learning*. PhD thesis, Univ. of Pennsylvania, 1993.

[6] C. Carpineto et G. Romamo. Order-Theoretical Ranking. *Journal of the American Society for Information Science*, 51(7), 2000.

[7] A. Cordier, J. Lieber, P. Molli, E. Nauer, H. Skaf-Molli, et Y. Toussaint. WIKI-TAAABLE : A semantic wiki as a blackboard for a textual case-based reasoning system. In *4th Workshop on Semantic Wikis (SemWiki2009), 6th European Semantic Web Conference*, Heraklion, 2009.

[8] B. Díaz-Agudo, P. Gervás, et P. A. González-Calero. Adaptation Guided Retrieval Based on Formal Concept Analysis. In *Case-Based Reasoning Research and Development : Proceedings of the Fifth International Conference on Case-Based Reasoning, ICCBR-03*, Lecture Notes in Artificial Intelligence 2689. Springer, 2003.

[9] B. Díaz-Agudo et P. A. González-Calero. Classification Based Retrieval Using Formal Concept Analysis. In D. W. Aha et I. Watson, editors, *Case-Based Reasoning Research and Development — Fourth International Conference on Case-Based Reasoning (ICCBR-01)*, Lecture Notes in Artificial Intelligence 2080, pages 173–188, 2001.

[10] B. Ganter et R Wille. *Formal Concept Analysis*. Springer, Heidelberg, 1999.

[11] P. Gervás, R. Hervás, et J.A. Recio-García. The Role of Natural Language Generation During Adaptation in Textual CBR, 2007.

[12] L.H. Ihrig et S. Kambhampati. Derivation replay for partial-order planning. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, 1994.

[13] L. Lamontagne, R. Bentebibel, E. Miry, et S. Despres. Finding Lexical Relationships for the Reuse of Investigation Reports. In *4th Workshop on Textual Case-Based Reasoning : Beyond Retrieval, ICCBR07*, 2007.

[14] L. Lamontagne et G. Lapalme. Textual reuse for email response. In *Advances in Case-Based Reasoning*, pages 234–246, Heidelberg, 2004. Springer.

[15] M.P. Marcus, B. Santorini, et M.A. Marcinkiewicz. Building a large annotated corpus of English : The Penn Treebank. *Computational linguistics*, 19(2) :313–330, 1994.

[16] N. Messai, M.-D. Devignes, A. Napoli, et M. Smaïl-Tabbone. Querying a Bioinformatic Data Sources Registry with Concept Lattices. In F. Dau et G. Mugnier, M.-L. aud Stumme, editors, *ICCS*, Heidelberg, 2005. Springer.

[17] A. Mille. *Associer expertise et expérience pour assister les tâches de l'utilisateur*. Habilitation à diriger des recherches, Université Claude Bernard, Lyon 1, 1998.

[18] H. Muñoz Avila et F. Weberskirch. Planning for manufacturing workpieces by storing, indexing and replaying planning decisions. In *Proceedings of the 3rd International Conference on AI Planning Systems, AIPS*, Edinburgh, 1996. AAAI Press.

[19] B. Smyth et M. T. Keane. Using adaptation knowledge to retrieve and adapt design cases. *Knowledge-Based Systems*, 9(2) :127–135, 1996.

[20] M. M. Veloso. *Planning and Learning by Analogical Reasoning*, volume 886 of *LNAI*. Springer, Heidelberg, 1994.

Cas socio-spatiaux pour le diagnostic prospectif de territoires

Sylvie Lardon [1], Florence Le Ber[2]
[1]INRA-SAD et AgroParisTech, UMR Métafort, Clermont-Ferrand
sylvie.lardon@engref.agroparistech.fr
[2]ENGEES LHYGES, Strasbourg et LORIA Nancy
florence.leber@engees.unistra.fr

Résumé

Les diagnostics prospectifs de territoires font appel a une expertise complexe et localisée. Or la méthodologie utilisée fait apparaître des ressemblances entre situations qui permettent d'accélérer le diagnostic, pour les personnes ayant une certaine expérience. Cette situation est donc propice au développement d'un système de raisonnement à partir de cas. Nous présentons ici en particulier le travail effectué pour le diagnostic de projets de pôles d'excellence rurale (PER), diagnostic qui s'appuie sur des schémas graphiques, appelés configuration socio-spatiale, et synthétisant l'information issue des dossiers décrivant les projets de ces territoires. Nous proposons de construire un référentiel pour ces PER, référentiel qui inclurait un système de raisonnement à partir de cas socio-spatiaux dont nous esquissons ici les principaux éléments.

Mots clés : diagnostic prospectif, configuration socio-spatiale, territoire, raisonnement à partir de cas, raisonnement spatial, graphes conceptuels

1 Introduction

De nombreux travaux portent sur des diagnostics et des prospectives de territoire, qu'ils soient élaborés par des praticiens, en particulier bureaux d'étude, pour les collectivités territoriales, ou par des étudiants dans le cadre de leur formation au développement et à l'aménagement des territoires. Or, chaque territoire étant spécifique, il manque un référentiel commun pour aider à raisonner localement en valorisant les expériences issues d'autres territoires. Construire un tel référentiel nécessite de représenter et d'exploiter de manière automatique les informations et connaissances élaborées lors de projets de territoires, informations et connaissances qui sont le plus souvent contenues dans des documents textuels, mais aussi dans des documents graphiques.

En effet, les représentations graphiques d'un territoire sont des éléments importants pour comprendre les enjeux de ce territoire et établir les stratégies d'action [8]. L'usage de représentations graphiques dans les démarches participatives de conception de projets de territoire accompagne la mise en forme des données spatiales d'une *mise en mots* : les modèles graphiques élaborés sont le support d'un énoncé textuel oral, explicitant les connaissances des acteurs. C'est pourquoi ces représentations graphiques des territoires doivent être au cœur d'un référentiel commun sur le diagnostic et la prospective territoriale.

Nous projetons de développer un tel référentiel pour caractériser les dynamiques de projets portés par les territoires et leur potentiel de développement. Nous nous appuyons sur l'exemple des pôles d'excellence rurale (notés PER) et proposons de mettre en œuvre une double approche :

1. d'extraction d'informations à partir de textes afin de construire des représentations graphiques incluant les aspects sociaux et spatiaux des projets de territoires décrits dans les dossiers ;
2. de raisonnement sur ces représentations, dans le but d'aider au diagnostic de nouveaux dossiers.

Les deux approches seront intégrées dans un même système, qui permettra à l'utilisateur d'accéder aux dossiers, documents et bases de connaissances liées. Le corpus dont nous disposons s'élève à 700 dossiers dont une trentaine ont fait l'objet d'une analyse approfondie.

Dans la suite de l'article, nous nous focalisons sur la deuxième approche, faisant appel au modèle du raisonnemet à partir de cas. Nous présenterons tout d'abord rapidement les principes du diagnostic prospectif de

territoires. Nous détaillerons ensuite la méthode et les produits du diagnostic appliqué aux dossiers de pôles d'excellence rurale avec un exemple dans la région du Livradois-Forez. Enfin nous présentons l'esquisse d'un système de raisonnement à partir de cas pour le diagnostic de ces dossiers, avant de conclure.

2 Diagnostic et prospective de territoires

2.1 Généralités

Le diagnostic de territoire doit permettre la formulation d'un jugement sur la cohérence du territoire, mais aussi sur la mobilisation des acteurs. Il accompagne un changement dans le comportement des acteurs et dans les transformations de l'espace, dans une perspective de développement territorial [18]. Le diagnostic de territoire est un moment privilégié de construction d'une vision commune du territoire. La prospective territoriale prend la forme de scénarios d'évolution pour mettre les acteurs en situation d'agir collectivement et d'anticiper les changements. Elle apporte un décalage qui facilite la mobilisation des acteurs et contribue à la construction d'un accord sur une vision de l'avenir.

L'itinéraire méthodologique de diagnostic prospectif de territoires est une façon d'articuler différentes méthodes de traitement des informations, mais aussi de mobilisation des acteurs, autour de la production de représentations spatiales, pour construire progressivement une vision partagée et stratégique du territoire [5]. La filiation des représentations spatiales au cours de cet itinéraire méthodologique rend compte de la construction du raisonnement, des points de passage obligés et des chemins alternatifs possibles. L'itinéraire comprend l'ossature et le principe d'agencement d'une démarche modulaire. Il constitue ainsi un guide pour anticiper l'adaptation des méthodes à différentes situations.

L'itinéraire méthodologique mobilise différentes sources d'informations, des « données froides » correspondant à des informations objectives, aux « données chaudes » issues de dires d'acteurs. Il confronte ces différentes informations et les met en perspective dans une vision globale du territoire. Il consiste en une décomposition-recomposition des principales structures et dynamiques du territoire, pour élaborer des scénarios d'évolution. La restitution aux acteurs met en évidence les enjeux du territoire et met en débat les choix stratégiques et propositions d'actions.

2.2 Application aux pôles d'excellence rurale

Les projets de pôles d'excellence rurale (PER) répondent à un dispositif lancé à l'initiative de la DATAR (Délégation interministérielle à l'aménagement du territoire et à l'attractivité régionale) pour financer des projets d'investissement dans des territoires ruraux dont les projets étaient reconnus d'excellence. Les territoires devaient pour cela présenter un dossier décrivant les principales caractéristiques de leur projet[1].

Il y a eu 700 dossiers de projets PER élaborés par des territoires (telles que communautés de communes, pays, parcs naturels régionaux ou départements) élaborés en 2006, dont 350 labellisés PER par la DATAR. Une nouvelle vague est lancée en 2010, avec une présentation des dossiers encore plus normalisée.

Nous avons formalisé un modèle d'analyse des PER [6] à partir du travail effectué sur 21 dossiers de la région Auvergne [17]. Ce modèle rend compte des caractéristiques d'organisation spatiale et sociale des actions prévues dans un PER et des réseaux d'acteurs concernés par la mise en œuvre de ces actions. Plus précisément, nous construisons des *configurations socio-spatiales* qui sont des représentations graphiques positionnant les principaux objets géographiques du territoire, ainsi que les actions localisées, prévues dans le PER et les acteurs concernés, à l'intérieur ou à l'extérieur du territoire de projet. Les liens entre acteurs et actions sont explicités. L'ensemble s'appuie sur une typologie des acteurs et de leur répartition dans l'espace.

Les configurations socio-spatiales sont établies à partir des chorèmes de territoires qui représentent les principales structures spatiales (villes, routes, montagnes, ...) et les principales dynamiques spatiales d'un territoire. Les chorèmes de territoires sont eux-mêmes établis à partir de données existantes sur le territoire (données cartographiques et statistiques) ou accessibles par enquêtes rapides auprès d'informateurs privilégiés. La représentation chorématique constitue un langage de description et d'interprétation des dynamiques en cours

[1] http://poles-excellence-rurale.datar.gouv.fr/

dans le territoire et donne à voir les enjeux auxquels les territoires sont confrontés (processus d'urbanisation, développement économique, attractivité touristique, préservation de l'environnement, ...).

À partir de ces configurations socio-spatiales et de l'historique de la constitution du PER, en particulier des acteurs ayant contribué à la construction du projet, il est possible de caractériser un potentiel de développement territorial porté par ce projet, en lien avec les grandes dynamiques d'évolution du territoire. Il s'agit en quelque sorte d'élaborer un diagnostic prospectif en confrontant les configurations socio-spatiales aux enjeux du territoire.

Ce modèle peut être testé sur d'autres dossiers PER pour lesquels nous disposons d'une « vérité terrain », dans la mesure où ils ont été étudiés par l'une des dix équipes de recherche impliquées dans la recherche évaluative des PER de la DATAR [6].

3 Un exemple de projet PER et sa configuration socio-spatiale

Nous détaillons ci-dessous un exemple de projet, celui du pôle d'excellence rurale du Parc Naturel Régional du Livradois-Forez qui repose sur une double valorisation, de la ressource forestière pour le bois-énergie et de la voie ferrée, initialement utilisée pour le développement touristique d'un train panoramique (figure 1).

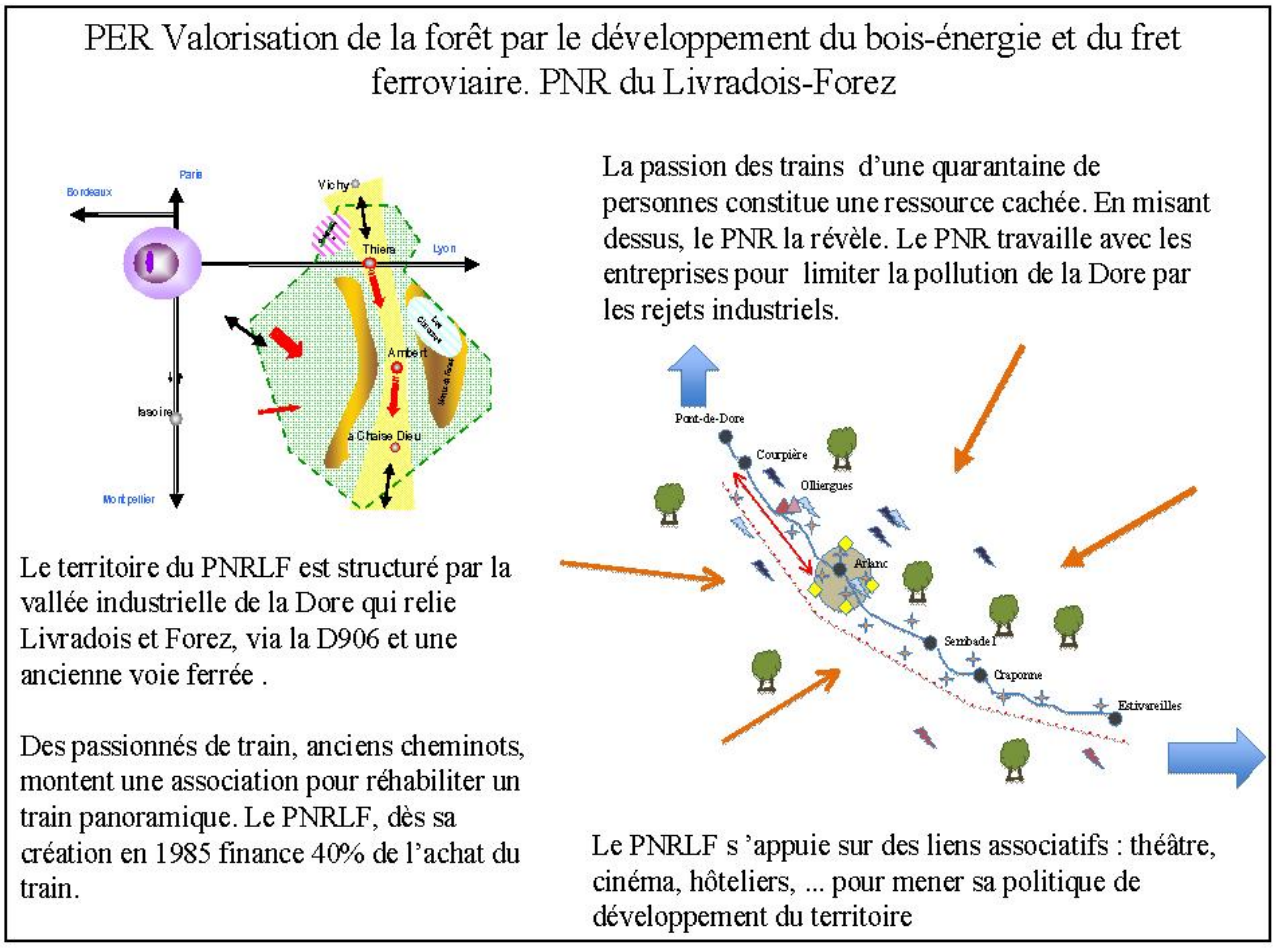


FIG. 1 – Éléments pour le diagnostic de territoire du PER du Parc Naturel Régional du Livradois-Forez.

Le projet de PER repose sur un diagnostic de territoire mettant en avant différents enjeux :

– développement de la filière bois-énergie pour valoriser les ressources locales (bois) et développer une filière d'énergie renouvelable (énergie bois, dans un contexte de changement climatique et de nécessité de réduction des gaz à effet de serre) et lutter contre une forte enforestation (plantation de résineux depuis une cinquantaine d'année qui ferment complètement le paysage) ;

– développement touristique pour assurer la multi-fonctionnalité d'un territoire de montagne soumis au déclin des activités anciennes (industrie le long de la vallée de la Dore, pastoralisme sur les chaumes) ;
– développement d'une économie résidentielle liée à la proximité de Clermont-Ferrand et maitrise des déplacements et mobilités.

La voie ferrée est impliquée sous ces trois aspects : pour le fret de marchandise, comme activité touristique et comme moyen potentiel de tranport "doux" (par opposition au "tout voiture").

L'étude du dossier conduit donc à construire une configuration socio-spatiale (figure 2) de lecture assez complexe, qui est explicitée ci-dessous. Le PER est porté par le Parc Naturel Régional du Livradois-Forez (PNRLF). Il est structuré par la voie ferrée qui relie les différentes communes le long de la vallée de la Dore, où sont situés les sièges des communautés de communes (dans lesquelles seront implantées des chaudières) et les sièges des entreprises (scieries, papeterie) et Clermont-Ferrand, siège de nombreuses institutions et organismes. Les objets spatiaux retenus sont ici la voie ferrée et les communes siège des entreprises et lieu d'installation des chaudières à bois.

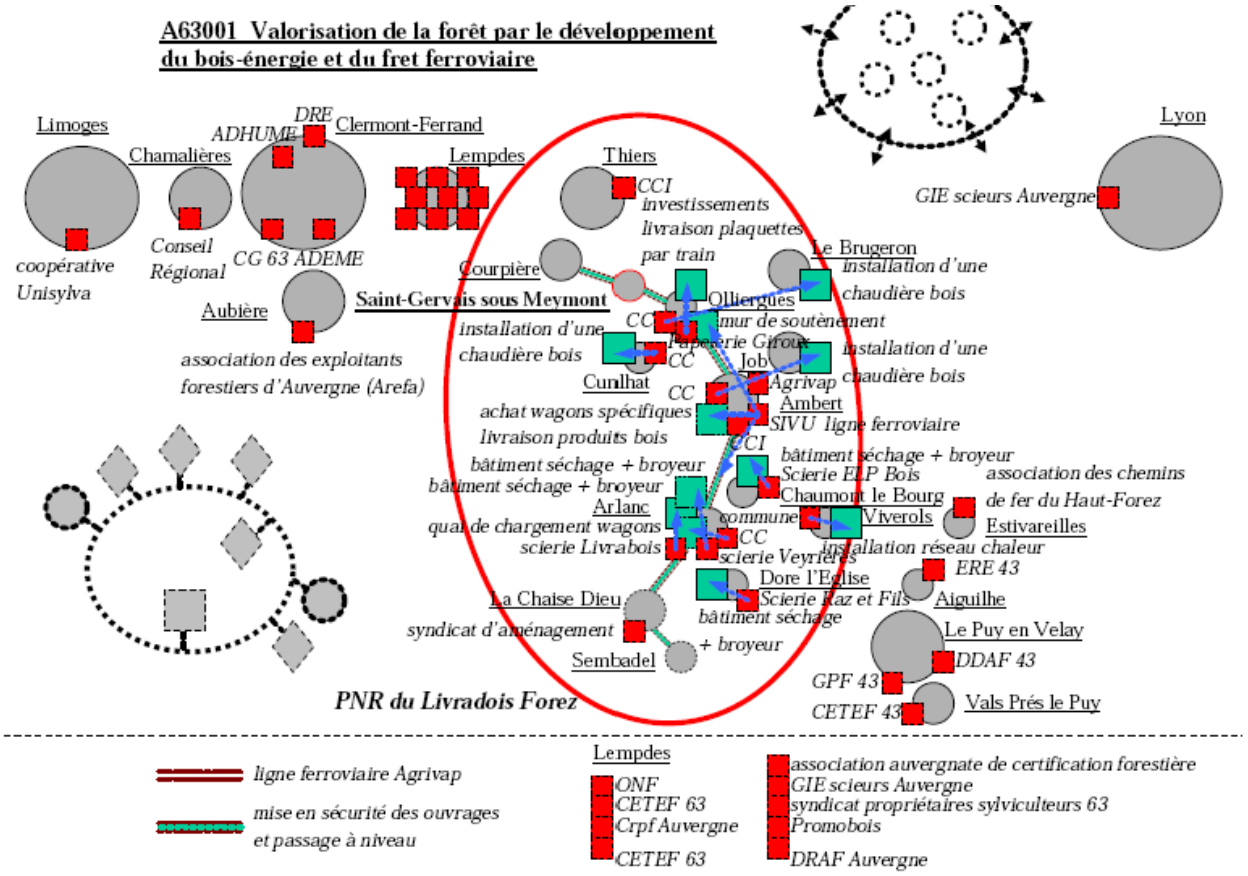


FIG. 2 – La configuration socio-spatiale du PER du Parc Naturel Régional du Livradois-Forez [17].

Les actions portées par le PER sont relatives à l'accroissement de la disponibilité du bois-énergie, l'accroissement de la demande en bois-énergie et l'acheminement des produits bois-énergie par le rail. Quatorze opérations ont été développées pour remplir ces objectifs :

– quatre opérations de financement de bâtiments de séchage de plaquettes destinées aux scieries du territoire proche de la voie ferrée et du bourg d'Arlanc : scierie Veyrière (et équipement broyeur), Raz & Fils, scierie ELP et scierie Ducros.
– cinq opérations de financement destinées aux équipements collectifs par la mise en place de chaudières à bois : au centre omnisports de Cunlhat ainsi qu'au centre d'hébergement du Brugeron, dans l'immeuble collectif de Job, la mise en place d'un réseau de chaleur collectif dans la commune de Viverols et dans la commune de Saint-Germain l'Herm.

– deux opérations de financement destinées aux papeteries de Giroux pour la construction d'un mur de soutènement, et pour la mise en sécurités des ouvrages et des passages à niveau.

Les acteurs concernés sont divers. Le Pôle d'Excellence Rurale est porté par le Parc, mais c'est au total une quarantaine d'acteurs – privés et publics – qui est directement et indirectement liée au projet PER. Les acteurs publics sont les quatorze établissements publics de coopération intercommunale (syndicats intercommunals, communautés de communes, divers syndicats), les Conseils généraux du Puy-de-Dôme et de la Haute-Loire, le Conseil régional d'Auvergne et le Parc Naturel Régional du Livradois-Forez. Au sein du PER, les acteurs moteurs du projet s'inscrivent dans le territoire local. Hormis le Parc qui est à l'initiative du projet PER, les acteurs clés pour la réalisation des actions sont incarnés par l'association AGRIVAP et les acteurs entrepreneuriaux tels qu'ABC et les papeteries Giroux. ABC est un acteur à la fois interne et externe à la dynamique locale (son siège social est à Clermont-Ferrand). Les autres acteurs externes favorisant le maintien des actions du PER sont la Région, les Départements du Puy-de-Dôme et de Haute-Loire. Ces différents acteurs entretiennent des relations différenciées.

Après l'analyse des documents et la synthèse sous forme de configuration socio-spatiale, le diagnostic est établi en répondant – entre autres – aux questions suivantes :

– Est-ce que tous les acteurs concernés par les thématiques du projet sont présents et sont en lien (par exemple, le PNR et les entreprises du bois n'entretiennent que peu de liens en dépit de la politique de valorisation de ressources du territoire, les entreprises sont sur un marché concurrentiel) ?
– Est-ce que tous les objets concernés par le projet sont pilotés par des acteurs, ou y a-t-il des objets orphelins (par exemple, rien n'est dit sur la production de la ressource forestière).
– Y a-t-il des acteurs émergents et de nouvelles relations ? par exemple, la société ABC, qui assure l'approvisionnement régulier en bois pour les chaudières grâce à la mise en commun de la production de plusieurs scieries, a été créée en cours de projet.

Cette procédure met en exergue ce que nous appelons des triplets socio-spatiaux, triplets composés d'un groupe d'acteurs, d'une (ou plusieurs) actions et d'un objet spatial support, le tout localisé, et où les acteurs entretiennent des liens entre eux et avec les actions. Notre hypothèse est que la configuration socio-spatiale peut se décomposer en triplets auxquels ont peut associer des éléments de diagnostic. Nous allons l'expliciter dans la prochaine partie.

4 Raisonner à partir de cas socio-spatiaux

Notre objectif est de constituer un référentiel de diagnostic et de prospective sur des territoires, de façon à aborder plus rapidement les enjeux de développement des territoires et de proposer des leviers d'action. On s'appuie pour cela sur l'exemple des projets de pôles d'excellence rurale en Auvergne. L'idée est de développer un outil d'aide à l'analyse des dossiers PER, pour caractériser le potentiel de développement des projets, sans réaliser un diagnostic complet tel que décrit en partie 2. Pour certains territoires bien connus, l'analyse du dossier PER a été effectuée complètement (en utilisant toutes les informations disponibles, y compris des enquêtes de terrain auprès des acteurs). Pour d'autres territoires, moins connus, on voudrait ne pas avoir à rechercher des informations complexes, mais pouvoir adapter les diagnostics déjà effectués en s'appuyant sur des ressemblances entre les projets de PER.

Le modèle du raisonnement à partir de cas [19, 20] se prête bien, *a priori*, à cet objectif. Plusieurs travaux récents ont d'ailleurs lié raisonnement à partir de cas et problèmes à caractéristiques spatiales, par exemple pour améliorer les services d'informations spatiales sur le web [15], ou pour la prédiction d'événements en territoire hostile [10]. D'autres travaux plus anciens avaient déjà souligné l'intérêt du raisonnement à partir de cas pour l'analyse de l'information géographique, avec un exemple en classification des sols [3, 4]. Nous mêmes avions souligné l'interêt de cette approche dans le cadre de l'agronomie des territoires, afin d'accompagner la démarche d'enquêtes et d'analyses menée par les agronomes [16]. L'idée, pour le projet décrit ici, est alors de tester la faisabilité et l'intérêt de constituer une base de cas de projets PER permettant de valoriser les expériences conduites dans les territoires et de formaliser les diagnostics et prospectives réalisés. L'exploitation de cette base – dans le cadre d'un système de raisonnement à partir de cas – pourrait permettre d'aider à l'élaboration ou au diagnostic de nouveaux projets de territoire.

Dans un premier temps, nous nous appuierons sur les configurations socio-spatiales construites et sur leur décomposition en triplets (acteurs, action, objets). Un *cas PER* serait donc constitué d'un ensemble de sous-cas (qu'on peut aussi considérer comme des index), les triplets auxquels on associe un élément de diagnostic (sur le rôle ou le potentiel de ce triplet dans le PER). Cette approche découle de travaux précédents, où nous avons développé un outil de représentation et de remémoration de « cas spatiaux », qui modélisaient des organisations spatiales d'exploitations agricoles. Ces cas étaient représentés sous forme de graphes bi-partites – similaires à des graphes conceptuels [13] – et implantés dans l'outil de logiques de description RACER [2]. Chaque cas d'exploitation était décomposé en sous-cas, représentant une structure spatiale assortie d'une explication (figure 3). La remémoration s'appuyait sur une ontologie de forme hiérarchique et l'outil permettait d'associer à une exploitation agricole à diagnostiquer un ensemble de sous-structures provenant d'autres exploitations déjà diagnostiquées [12, 7].

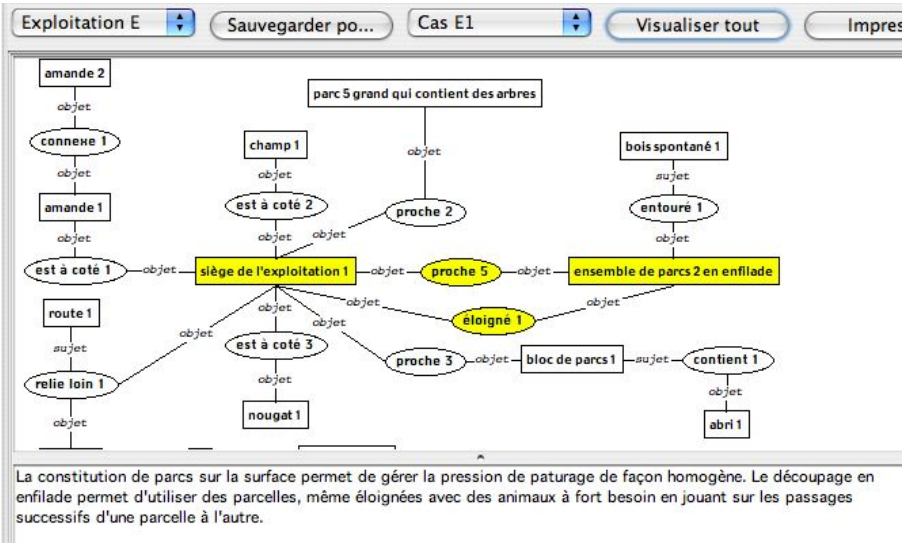


FIG. 3 – Graphe représentant une structure spatiale d'exploitation : la sous-structure marquée en jaune est assortie d'une explication, en bas de la figure [11].

Pour les PER, nous envisageons d'utiliser également des graphes conceptuels pour représenter et manipuler les cas, mais en combinant plusieurs types de connaissances, puisqu'il s'agit de cas socio-spatiaux. Un cas sera donc constitué d'un triplet (acteurs, actions, objets spatiaux), complété d'un diagnostic, le triplet constituant la partie « problème » du cas et le diagnostic la partie « solution ». Dans l'exemple traité ci-dessus, on aurait par exemple les cas suivants :

- Cas 1 : Triplet = (Société ABC/conseil municipal - équipement - chaudière bois), Diagnostic = (sécurise la demande en bois énergie par les collectivités territoriales)
- Cas 2 : Triplet = (Scieurs - équipement- bâtiment de séchage), Diagnostic = (produit des plaquettes pour alimenter le marché)
- Cas 3 : Triplet = (Société ABC/scieurs - approvisionnement - bois), Diagnostic = (sécurise l'offre de bois sur le marché)

Ces cas sont rattachés à un contexte général (ici : développement d'une énergie renouvelable, le chauffage au bois) et à la configuration socio-spatiale dont ils sont issus (PER du Parc Naturel Régional du Livradois-Forez, cf. figure 2). Une première ébauche de formalisation est en cours à l'aide du logiciel COGITANT[2] qui permet de représenter concepts, relations, faits et règles puis d'effectuer des requêtes.

Le modèle de raisonnement à partir de cas sur les PER pourrait alors fonctionner ainsi : soit un PER-cible (à évaluer) décrit par un ensemble de graphes-triplets, extraits d'un dossier. Dans l'étape de remémoration, ces triplets sont appariés avec des triplets sources de la base de cas qui leur sont similaires. Les diagnostics, détaillant les rôles ou potentiels des triplets sources, pourront alors être reportés ou adaptés aux triplets cibles (étape d'adaptation). Le calcul de similarité – puis l'adaptation – s'appuient sur les connaissances du domaine,

[2] http://cogitant.sourceforge.net/

par exemple les types d'acteurs (institutionnels, entrepreneuriaux, ...) ou les formes de relations entre acteurs. Notons que des éléments d'une base de connaissances sont déjà disponibles grâce à la formalisation du domaine menée pour l'établissement de configurations socio-spatiales dans le diagnostic des dossiers PER [17]. En revanche nous avons peu d'éléments sur les connaissances d'adaptation et la place de l'utilisateur-expert sera donc centrale dans les étapes d'adaptation et d'évaluation. Dans un premier temps et pour pouvoir construire les connaissances d'adaptation, l'adaptation sera donc manuelle.

Conclusion et Perspectives

Cet article rend compte de réflexions préliminaires au développement d'un système d'aide à l'analyse et à l'évaluation de dossiers de Pôles d'Excellence Rurale. L'idée générale est de constituer un référentiel pour ces pôles, qui soit à la fois une ressource pour construire de tels projets et pour en faire le diagnostic. Ce référentiel s'appuierait d'une part sur un outil d'extraction automatique d'informations spatiales à partir de textes comme proposé dans [14], d'autre part sur un module de raisonnement à partir de cas socio-spatiaux, module dont nous avons présenté le principe ici. L'ensemble du référentiel devrait permettre d'accéder aux dossiers et de générer ou manipuler les différents éléments de diagnostic, chorèmes, configurations socio-spatiales, graphes-triplets.

Pour développer le module de raisonnement à partir de cas nous devons mettre en œuvre un processus d'acquisition de connaissances, à la fois pour extraire et formaliser les triplets (acteurs, actions, objets spatiaux) et leur diagnostic et pour formaliser les connaissances utiles à la remémoration et à l'adaptation. Ces connaissances sont largement d'ordre qualitatif, ce qui nous conduira certainement à faire appel aux modèles qualitatifs du temps et de l'espace [21, 1, 9].

Références

[1] M. Aiello, I. Pratt-Hartmann, et J. van Benthem, editors. *Handbook of Spatial Logics*. Springer, 2007.

[2] V. Haarslev, R. Möller, et A.-Y. Turhan. Racer user's guide and reference manual. Université de Hambourg, février 2001.

[3] A. Holt et G.L. Benwell. Case-Based Reasoning and Spatial Analysis. *Journal of the Urban and Regional Information Systems Association*, 8 :27–36, 1996.

[4] A. Holt et G.L. Benwell. Applying case-based reasoning techniques in GIS. *International Journal of Geographical Information Science*, 13(1) :9–25, 1999.

[5] S. Lardon. Analyse spatiale pour le diagnostic et le projet de territoire. In S. Lardon, P. Moquay, et Y. Poss, editors, *Développement territorial et diagnostic prospectif. Réflexions autour du viaduc de Millau*, Essai, pages 169–191. Éditions de l'Aube, 2007.

[6] S. Lardon et P. Cayre. Les pôles d'excellence rurale : de nouveaux modèles de développement pour les territoires ? Rapport final recherches évaluatives de la DATAR, 2009.

[7] S. Lardon, F. Le Ber, J.-L. Metzger, et P.-L. Osty. Une démarche et un outil pour modéliser et comparer l'organisation spatiale d'exploitations agricoles. *Revue internationale de Géomatique*, 15(3) :263–280, 2005.

[8] S. Lardon et V. Piveteau. Méthodologie de diagnostic pour le projet de territoire : une approche par les modèles spatiaux. *Géocarrefour*, 2005.

[9] F. Le Ber, G. Ligozat, et O. Papini, editors. *Raisonnements sur l'espace et le temps : des modèles aux applications*. Traité IGAT - Géomatique. Lavoisier, Paris, 2007.

[10] H. Li, H. Muñoz-Avila, D. Bransen, C. Hogg, et R. Alonso. Spatial event prediction by combining value function approximation and case-based reasoning. In L. McGinty et D.C. Wilson, editors, *ICCBR 2009*, LNAI 5650, pages 465–478. Springer-Verlag, 2009.

[11] J.-L. Metzger. Contribution à l'élaboration d'un modèle de raisonnement à partir de cas pour l'aide à l'interprétation d'organisations spatiales agricoles. Thèse de doctorat en informatique, Université Henri Poincaré Nancy 1, avril 2005.

[12] J.-L. Metzger, F. Le Ber, et A. Napoli. Éléments pour la modélisation et la représentation de structures spatiales agricoles. In *Actes de la conférence Langages et Modèles à Objets (LMO'2003), Vannes*, RSTI L'objet 9/2003, pages 197–210. Hermès Lavoisier, janvier 2003.

[13] M.-L. Mugnier et M. Chein. Représenter des connaissances et raisonner avec des graphes. *Revue d'Intelligence Artificielle*, 10(1) :7–56, 1996. Numéro spécial graphes conceptuels.

[14] V.T. Nguyen, M. Gaio, et C. Sallaberry. Recherche de relations spatio-temporelles : une méthode basée sur l'analyse de corpus textuels. In *TIA 2009 : 8th International Conference on Terminology and Artificial Intelligence TIA'09WS : Acquisition et modélisation de relations sémantiques*, pages 1–6, Toulouse, France, novembre 2009.

[15] T. Osman, D. Thakker, Y. Yang, et C. Claramunt. Semantic spatial web services with case-based reasoning. In *Web and Wireless Geographical Information Systems, Proceedings of the 6th International Symposium, W2GIS 2006*, LNCS 4295, pages 247–258, Hong Kong, China, décembre 2006. Springer.

[16] P.-L. Osty, F. Le Ber, et J. Lieber. Raisonnement à partir de cas et agronomie des territoires – constructions croisées. *Revue d'Anthropologie des Connaissances*, 2(2) :169–193, 2008.

[17] R. Perraud. Les configurations socio-spatiales et temporelles des Pôles d'Excellence Rurale : itinéraire méthodologique et construction d'archétypes à partir des 21 PER d'Auvergne. Mémoire de Master DTNR, 2009. 87 pages + annexes.

[18] V. Piveteau et S. Lardon. Chorèmes et diagnostics de territoire : une expérience de formation. *Mappemonde*, 2002.

[19] C. K. Riesbeck et R. C. Schank. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Inc, Hillsdale, New Jersey, 1989.

[20] R. C. Schank, A. Kass, et C. K. Riesbeck, editors. *Inside Case-Based Explanation*. Lawrence Erlbaum Associates, Inc, Hillsdale, New Jersey, 1994.

[21] O. Stock, editor. *Spatial and Temporal Reasoning*. Kluwer Academic Publishers, 1997.

Providing assistance by reusing episodes stored in traces: a case study with SAP Business Objects Explorer [1]

Raafat ZARKA[1], Amélie CORDIER[2], Francoise CORVAISIER[3], Alain MILLE[2]
[1]SAP Business Objects, INSA de Lyon, LIRIS, raafat.zarka@sap.com
[2]Université Lyon 1, LIRIS, {prenom.nom}@liris.cnrs.fr
[3]SAP Business Objects, francoise.corvaisier@sap.com

Abstract

It is a big challenge in the information technology field to develop techniques to help users in their tasks. For that purpose, we need to develop assistants able to help people without disturbing them in their main task. This paper discusses building an assistant based on traces and making use of a trace-based system. Traces can be used as a knowledge source to discover other useful knowledge but also to reuse experience. The main idea of the approach is to find useful episodes (previous experiences) in interaction traces and to reuse them to provide users with contextualized help. The assistant can then adapt the retrieved episodes to provide assistance to the user by reusing previous experiences. This work has been done in partnership with SAP-BO. We have implemented our proposal in the SAP explorer project which aims to ensure that all business users have easy access to all the information they need to make confident decisions based on up-to-date, reliable information, so they need to help business users while doing their tasks. The main contribution described in this paper is the algorithm enabling us to retrieve past episodes corresponding to a "task signature" (a description of the main characteristics of the episode). For that purpose, we have implemented a solution based on Finite State Machines.

Keywords: Assistant agent, Trace-Based Reasoning, similarity measure, task signature, Finite State Machine, sequence pattern mining.

1 Introduction

Everyday new software products appear in different domains, and a major challenge is to make their usage as easy as possible. This challenge leads us to focus on various ways to develop intelligent applications more flexible and scalable. For example, there is a growing interest in developing auto-adaptable applications able to fit specific user needs, or able to provide them with a relevant assistance at the right time and the right place. This adaptability of application is a major concern for groups such as SAP. Indeed, for effective decision making, business users require quick, easy answers to off-the-cuff questions and a better understanding of the business, without extensive training and dependence on IT. This is the reason why they need reactive applications. It was also the main goal for SAP when developing SAP-BO Explorer software. SAP-BO Explorer is software offers users an intuitive path to quickly search and explore data for instant insight into their business [1]. But still, the application is hard to master for most users, and additional assistance would be useful. To help reaching this goal we work on developing assistant enabling users to explore the system easily, to reuse their previous experiences and to benefit from advance reasoning functionalities provided by the system.

This paper presents a preliminary research work, in the context of a Master Thesis. We propose to make use of the trace-based reasoning (TBR) paradigm in the development of an assistant for a given application. While using a system, a user leaves behind him "interaction traces". Because interaction traces capture users' experiences, they can be exploited to provide experience-based assistance thus helping the user to conduct his task efficiently. Hence, in TBR, interaction traces are used as a specific knowledge container. In this field, our contribution is to provide an algorithm to retrieve "episodes" (previous experiences) in a trace and to reuse them to provide assistance to users in a specific context. The episode is a sequence of interactions that allow solving a problem. The algorithm takes as input the "current" trace and makes use of a base of

[1] This work was supported by SAP Business Objects, 157-159, rue Anatole France, 92309 Levallois-Perret Cedex, France

"task signatures" and a base of traces. A task signature is a formal description of a specific task for which a user might need assistance. Task signatures are given by experts in the form of rules that are transformed in automata. The proposed algorithm processes this automaton. This work is supported by SAP Business Objects and we are working on a specific application, SAP-BO Explorer. We have instrumented the application to be able to collect interaction traces. The examples provided in this example come from the demo dataset of the application.

This paper is organized as follows. In section 2, we describe the context of this work and we do a short survey on trace-based systems as well as assistance systems. Section 3 presents the general architecture of the assistant and gives definitions and models required to understand our retrieval algorithm. Then, the main algorithm is described and the assistance principles are illustrated. Section 4 discusses our approach and the issues it raises. The paper ends with a conclusion in section 5.

2 State of the art

In computer science, traces have many shapes and usages; log files, navigation history, versioning. That is why many researchers were interested to study the benefits of using these traces to develop more powerful applications with the trace of interactions. The SILEX team[2] (Supporting Interaction and Learning by Experience) is interested in this topic and has proposed the notion of *M-Trace* (Modelled Trace) and of Trace Based System [2] .

There are various researches on traces, aiming at proposing models and tools for representing, transforming, sharing and visualizing traces of users' experiences. In [2] the authors present a new approach for modelling user's experience while using a computer system, with the objective of reusing this experience as knowledge in context for helping users along their tasks. This work has evolved, and recent approaches propose to handle traces using a Trace-Based System (TBS). A TBS can be seen as a kind of Knowledge-Based Systems (KBS) whose main source of knowledge is the set of traces representing user-system interactions and evolving with users' activities. In [3], the authors presented the general architecture of the TBS framework, and they specified a language to describe *M-Traces* (i.e. modelled traces), *M-Trace*s based patterns, queries, and transformations. The work presented on this paper uses this formalization.

While working with traces, one big challenge is to define trace models, because models should be defined according to the target application. That is why the trace theory has defined the concept of *M-Trace* (modelled trace), that defines the vocabulary of the trace obsels (a trace consists of a set of observed elements which has been abbreviated in obsels), the properties of the obsels, the relationships that may exist between these elements, and the way to timestamp each obsel. According to *M-Trace*, an episode is a sequence of obsels correspondent to the interactions that allow solving a problem. With the availability of *M-trace* concept and by using the Trace Based Management Systems (see section 3.1) new trace-based applications can be dynamically designed. For example, [4] designed a model for addressing issues related to traces modelling and visualization in synchronous collaborative learning activities, to allow users to visualize and analyse their own experiences by observing their actions on the learning system. They proposed a generic trace model for synchronous collaborative activity based on the notion of interaction in two modes: whiteboard sharing and text chatting. Another example is [5], where the authors presented an approach to context-aware assisting systems relying on traces of user interactions. They used the notion of Explained Task Signatures (ExTaSi) to describe tasks performed by users. The system collects traces and transforms them. The assistance process is triggered when the current trace matches the beginning of an ExTaSi. Then, the assisting agent searches the TBS for all the occurrences of this ExTaSi in the interaction trace. These occurrences can be ranked according to a similarity measure defined for the current trace. However, we may note that, according to the authors, the assistant agent requires some actions from the user to validate the tasks that make sense to him.

Another related work, described in [6], is about building systems that can ease the sharing and re-using of experience between large communities of users. The authors proposed to use traces of interactions as a new way of performing CBR (Case-Based Reasoning) to enhance the traditional CBR by providing more flexible ways of reasoning from experience. They showed that the traces can be used to record experiences

[2] http://liris.cnrs.fr/silex

"in context" and thus, bring more flexibility to traditional CBR. The agile CBR idea described by Susan Craw in [7] shares the same point of view, where the main idea of agile CBR is to transform traditional CBR into a "dynamic, knowledge-rich, self-organizing, cooperative problem-solving methodology". [6] has raised a lot of researches issues, especially for building Trace-Based Assistants (TBA). Some of them, such as "how to define a similarity measure", will be discussed in our contribution.

Most CBR systems try to solve problems by using log files. For example [8] presents a new scheme for coding web server log data into sessions of behavioural sequences. Where [9] propose a Ceaseless CBR model that can enrich the CBR diagram that considers the CBR task as on-going rather than one-shot, to support the analysis of unsegmented sequences of observational data stemming from multiple coincidental sources. A hybrid neuro-symbolic system has been presented in [10], this model takes into account the temporal dynamics contained in the sequences and allows to avoid problems related to the comparison of different length sequences. Another example is [11], where the authors presented case-based system for cooperative information browsing, called "Broadway". This system follows a group of users during their navigations on the WWW (proxy-based architecture) and advises them by displaying a list of potentially relevant documents to visit next. The advices are based mainly on similarity of ordered sequence of past accessed documents. While in our work we make use of traces on which we can do transformations to be able to have more abstract traces which is not possible with raw logs.

One major problem in trace-based reasoning is to retrieve episodes in traces. To tackle this issue, data mining techniques could be helpful. Sequential pattern mining was first introduced in [12] for sequential database that consists of sequence of ordered events. This algorithm aims at finding frequent occurrences of sequences allowing to describe the data or to predict future data. Time-related data mining is an active research area. In [13] we can find a comprehensive description of this domain. Existing methods only focus on the concept of frequency because of the assumption that sequences' behaviours do not change over time. The environment from which the data is generated is often dynamic, however, so the sequences' behaviours may change over time. To adapt the discovered patterns to these changes, new concepts are incorporated into traditional sequential pattern mining [14].

We aim to develop an assistant to help users doing their tasks effectively, but developing assistants is not an easy work; because sometimes they can have negative effects. For example, the Office Assistant drew a strongly negative response from many users; that is why Microsoft has removed the office assistant in the new versions of Microsoft Office and replaced it with a new online help system [15]. The Office Assistant was a feature to assist users by way of an interactive animated character, which was interfaced with the Office help content. It used technology initially from Microsoft Bob and later Microsoft Agent, offering advice based on Bayesian algorithms. It popped up when the program determined the user could be assisted with using Office wizards, searching help, or advising users on using Office features more effectively. It presented tips and keyboard shortcuts. For example, typing an address followed by "Dear" would cause Clip pit to pop up and say, "It looks like you're writing a letter. Would you like help?". According to some studies, even if the recommendation was sometimes helpful, in most of the situation, users considered it as annoying. Often, the problem with assistant is that they are designed to assist "prototypical users". However, assistance is a very specific task which is efficient when it takes into account the needs of specific users. This probably is the reason why a lot of assistant agents have failed. In the context of this work, we aim at designing auto-adaptative assistants able to take into account specific needs of users. We are aware that this goal involves a lot of other issues (ergonomics, human-computer interactions, etc.), but they are out of the scope of this paper.

3 Trace-Based Assistant: an approach based on FSM principle

In our paper we are interested in reusing experience of the user to help him achieving his tasks effectively. Suppose that the user is interacting with the system for doing a specific task, the assistant will automatically observe what the user is doing, then apply the same transformations on the past traces using tasks signatures to the current traces, while task signatures are already defined and stored in the task signatures base. After that, the assistant will retrieve and present to the user the past episodes that correspond to his current task, these episodes will be ranked using similarity measures, and adapted to the user to be able to reuse them.

In order to build our Trace-Based Assistant (TBA) we need to collect the traces and manipulate them by using a Trace-Based Management System. The assistant will be connected with, executing queries to find the related episodes or even to add new experiences. Figure 1 shows the general architecture of our trace-based system, we can see that there are three main parts, the system for which we want to provide the user with help, the TBMS that collects the traces of the user while he is working on the application and support many functionalities like transformation, querying, and visualizing by using the task signatures in the tasks base, and the assistant that observes the way the user interacts with the system, to help him doing his task effectively, by retrieving the related episodes from the TBMS and adapting them.

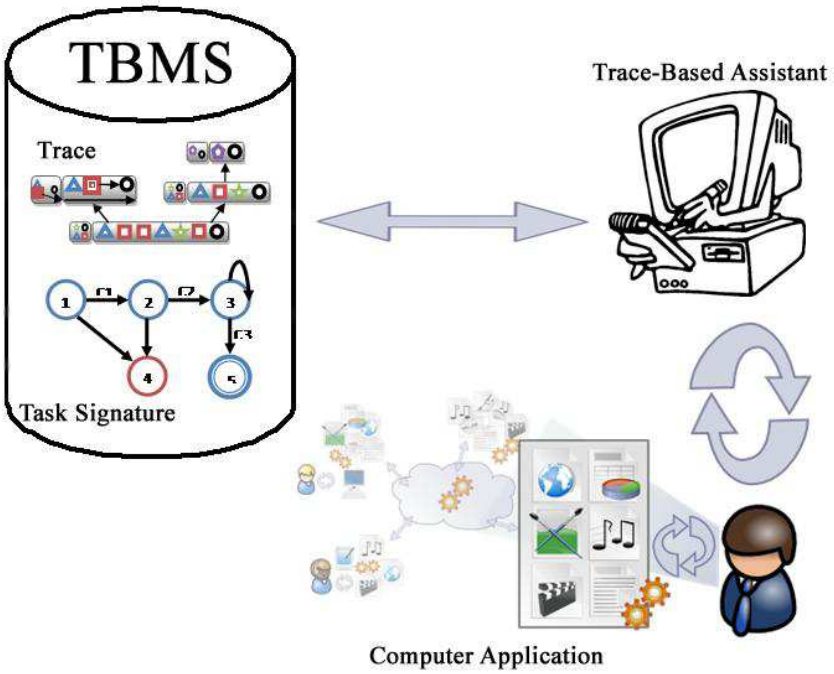


Figure 1: Trace-Based Assistant: a general architecture

3.1 Trace-Based Management System

We can see a TBMS as a Database Management System, by considering these mappings: Trace models / ERD model, obsels/data as the main objects in this system, transformation, querying, and visualization functionalities. [16] defined the notion of Trace Based Management System (TBMS) as systems devoted to the management of modelled traces. A modelled trace is a trace explicitly associated with its trace model. Each trace consists of a set of elements called obsels which results from the observation of the interaction between the user and the system, and each obsel has a set of attributes/values. A modelled trace formally defines the structure and the types of obsels, and the relationships between these obsels. It can be considered as a formal ontology that describes the vocabulary of the trace.

The various traces are managed by the Trace-Based Management System. The collecting process is the first function that builds the primary trace by observing and storing the interactions between the user (could be agent, user or external source) and the system. This primary trace can be transformed to generate a new trace of a new level. Transformation can be applying filters, rewriting and aggregating elements, computing elements attributes, etc. We can apply the transformation on the transformed traces and have many levels of abstraction, which could be more reusable and exploitable in a given context than the primary trace. A TBMS guarantees the possibility, at any time, to navigate between transformed traces. This ability provides an important flexibility. Figure 2 shows a primary trace *T1* which is transformed into *T2* and *T3* (level 2), and *T3* is transformed in its turn into *T4* (level 3). Each trace has its trace model that contains different types of obsels that may have relations between them. The trace *T2* is somewhat detailed where *M2* is its trace model that contains three obsel types (*c1, c2, c3*) and one relation type *r1*. *T2* contains three obsels (*o1, o2, o3*) and there is a relation between *o2* and *o3*.

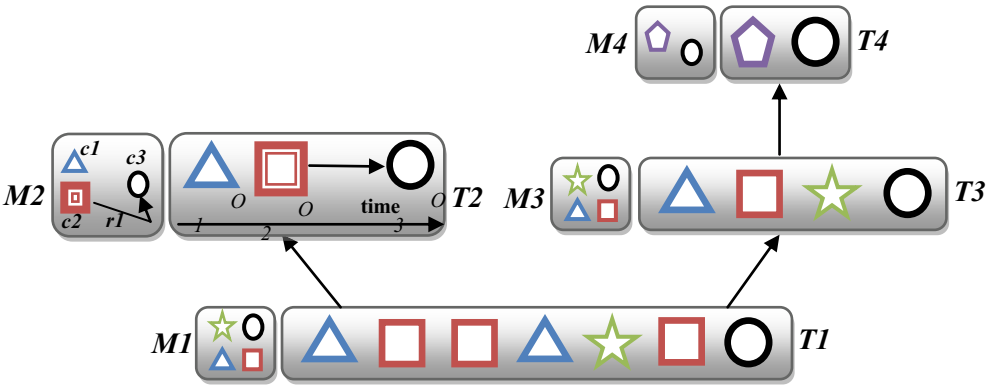


Figure 2: Exemple of M-Trace

TBMS provides us also with a querying service that is dedicated to retrieving traces from the trace base or to write or update the traces.

A complete formalization of the meta-model proposal for traces models, traces, queries and transformations can be found with precise semantics in [3]. The core of our work is building the Trace-Based Assistant that implements such meta-model.

SAP-BO Explorer offers to the users an intuitive path to quickly search and explore data for instant insight into their business. Firstly we modified this software for being able to collect the obsels. This application is divided into two sides, the server side which is implemented in Java, and the client side which is a flex application. Many users could work on this application at the same time. That is why we made the collecting process on the client side. Every time the user tries to use the system, a new session is opened. Each session contains many obsels, so each action of the user will be collected as an obsel presented in a XML format specifying the obsel type, timestamps, and the values of this obsel. We suppose that the interface of SAP-BO Explorer is divided into task oriented blocks, where each block contains obsels with similar kind of tasks. For example when the user tries to choose a measure, we capture this action as an obsel of the type “Select measure” from the second block “The measures block” and with a value “Quantity Sold” and the current timestamps.

Each session is presented as *M-Trace* stored in the form of XML. Each session has a unique ID, the user who did this session, and the temporal list of the obsels that happened in this session. When a user connects to SAP-BO Explorer, a request to the server-side will be sent to open a new session and creating a new XML output file for this session. Every time a new obsel is collected, it is presented in XML format and sent to the server to be added to the session file. The management of the session is in the server side, where we keep all the opened sessions in a Hash Table. When a session is closed it is removed from the Hash Table. So we don’t need to close and open the file every time we collect a new obsel, we just keep the files of the current sessions opened. If the same user opens two sessions at the same time, they are stored in the same file.

3.2 Task Signatures Base

Task signature concept has been introduced in [2] as a set of event declarations, entity declarations, relations, and temporal constraints. With the trace model and task signatures based on this trace model, it is possible to define new obsel types in the trace model by using the task signatures to generate a new trace by replacing the obsels of the task with one more abstract obsel.

A Finite State Machine (FSM) is a model of behaviour composed of a finite number of states, transitions between those states, and actions. It is similar to a "flow graph" where we can inspect the way in which the logic runs when certain conditions are met [17]. FSMs are widely used in modelling of application behaviour, design of hardware digital systems, software engineering, compilers, network protocols, and the study of computation and languages. Many researchers worked on modelling software with Finite State Machines [18], but as far as we know, there is no work in using FSM to do episode retrieval.

We will extend the concept of task signature to be able to define more sophisticated signatures rules, especially by adding more time constraints. Our extension will be to use a Finite State Machine as a sequence detector to produce a binary output saying either 'yes' or 'no' to answer whether an episode is accepted by the machine or not. At the time when all obsels of the current episode is processed, if the final state is an accept state so this episode is accepted and matched the task signature, otherwise it is rejected. We will not explain more about how FSM are working, but we just want to explain how we use them in our model to define the task signatures.

By using FSM we can apply any rule we want, like recognizing series of obsels which can be directly linked or separated. In Figure 3 we can see an example of a task signature saying that an episode to be accepted it should start by an obsel of type *c1* followed directly by at least one obsel of type *c2* and then an obsel of type *c3*, where state1 is the entry state, state5 is the end success state, and state4 is the failure state. When we want to check if an episode has a task signature, we can pass it to the FSM task signature, and see in which state it will ends. If it is the refuse state, it means that it doesn't has this signature, else if it is the accept state, it has this signature, else it is one of the intermediate states, here we can measure the distance between this state and the accept state to be used by the assistant for helping the user to reach the accept state and do his task. In addition to the FSM we could define temporal rules like the maximum duration of the task, the distance between two obsels, etc

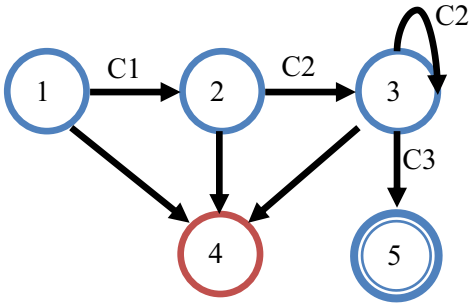


Figure 3: Example of a FSM Task Signature

As an example in SAP-BO Explorer: the task signature in Figure 3 could be "Visualizing the quantity sold in New York of the year 2001 and changing the chart type to pie chart" So the obsels will be:

- C1: Obsel (Type='Select Measure', value='Quantity Sold')
- C2: Obsel (Type='Choose Dimension Value', value='Year is 2010')
- C2: Obsel (Type='Choose Dimension Value', value='City is New York')
- C3: Obsel (Type='Change Chart Type', value='pie')

In this example the task starts by selecting the measure and then we can choose many dimension values without any conditions in their orders, but it should be followed by changing the chart type.

Task signature base is application oriented that will be manually filled by system experts, where we suppose that the experts are capable of defining the task signatures as the form of rules. Every rule is converted to a FSM as a form of Graph that will be used as an input to our retrieving algorithm, so we need a conversion process that takes a rule defined by system experts, and returns a graph representing this rule, the nodes of this graph will be the states, where the links will be the transitions.

We do the transformations of the traces using the task signatures stored in the tasks base. Firstly we search for all episodes in the primary trace that has a task signature and generate the trace of level 1 by replacing the obsels of the task signature by one new obsel describing this task with setting the start point as the start point of the first happened obsel, the end point as the end point of the last happened obsel. In case that there are some episodes have more than one task signature, we generate many transformed traces at the same level, so we can have many traces at the same level with differences in the obsels types. We repeat the same operation for all the traces of level k to generate traces of the level k+1, until we don't have anymore episodes with new task signatures. That will make the trace model in a form of a graph because we could have two childs have the same parent.

3.3 Trace-Based Assistant

The assistant is connected to a TBMS and has access to all its functionalities and knowledge, including traces and task signatures. During the interaction between the user and the system, the assistant generates the current online trace. This trace is transformed in the TBMS using the same task signatures that have been used for the past trace transformations. As the traces are time-series data, we can depend on the concept of sequence pattern mining [13] for developing the assistant to be able to retrieve the related episodes. (After the retrieving step we need to adapt the retrieved episodes to be reusable by the user.)

In our algorithm of retrieving the related episodes and ranking them (see Algorithm 1), either we consider that the task signature base is already filled by the signatures by the system experts manually, or we suppose that we are capable of generating task signatures automatically while the user is doing his task. The algorithm takes the current trace and its modelled trace, the task signature base as input, and it retrieves a ranked list of the suggested obsels. These obsels are the suggested tasks that the user can do after his current task. We can specify a maximum length of the sliding window for indicating the number of obsels in the current episode. The algorithm will start from the top level transformed traces, for each trace in the top level we take the current episode of the maximum sliding window length to find all episodes with the same length. If there are no suggested results, it will decrease the length and try again until the length of one obsel.

If the algorithm finishes without any results, it repeats all the process for each child trace (decreasing one level) until it arrives to the primary trace. In each iteration, it tries to retrieve all the episodes of the same level in the historical trace that have the same obsel types in the same order, and returns rules indicating their successor episode (the related episode) with two values (support and confidence). These values will be used to rank the results and avoid some results which are under specified thresholds. Support and confidence are similar to the same statistics measures of significance and interest used by Association Rules and Sequence Pattern Mining algorithms (See **Definition 1, Definition 2, Definition 3**).
After finishing the retrieving process, we need to adapt the retrieved obsels to fit the current task, because the values of the obsels could be different so we need to find the sufficient value of the solution obsel.

Definition 1: Rule

Let $M_{i,j}=\{c_1,c_2 , .. c_n\}$ be the modelled trace i of the level j that has a set of n obsel type.
Let $T_{i,j}=\{o_1, o_2, .. o_m\}$ be the trace i of the level j which is a sequence (time ordered list) of m obsel(s). Each obsel has a type $c_k \in M_{i,j}$, corresponding attribute values, and the associated time stamps. An episode is a sequence of $T_{i,j}$ satisfying a specific episode signature.
We can define a rule as an implication of the form $X \rightarrow Y$, where X,Y are episodes of $M_{i,j}$ called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule respectively.

In our assistant we will suppose that the length of the consequent episode is one obsel. To select interesting rules from the set of all possible rules and ranking them, constraints on various measures of significance, and interest can be used like minimum thresholds on support and confidence.

Definition 2: Support

The support supp(X) of an episode X is its occurrence percentage of all episodes with the same signature in the trace $T_{i,j}$.

$$\text{Supp}(X) = (\text{Number of occurrence of } X \times \text{length}(X)) / \text{length}(T_{i,j})$$

Definition 3: Confidence

The confidence of a rule is defined conf($X \rightarrow Y$) as the occurrence percentage of the episodes that have a signature of merging X, Y to the occurrence of the episodes that have the signature of X.

$$\text{Conf}(X \rightarrow Y) = \text{supp}(X+Y) / \text{supp}(X)$$
Where the operator + returns new episode by adding Y to the end of X.

Algorithm 1: TBA retrieving algorithm

```
Input: T, M (Where T is the current Trace, and M is the current modelled
trace)
Output: Suggested (a ranked list of the adapted suggested obsels)

Suggested=Ø
For each trace T_i,j in TopLevelTraces(T)
      Results = Retrieve (T_i,j, M_i,j)
      For each rule in Results
  AdaptedRule = Adapt (rule)
  Suggested.Add(AdaptedRule)
      End For
End For
Return Suggested

Retrieve Function:
Input: T_i,j, M_i,j (Where T_i,j is the current Trace j of level i, and M_i,j is
the current modelled trace j of level i)
Output: Results (a ranked list of the suggested obsels)

Results=Ø
Length = EpisodeLength;
While (Length > 0)
      Episode = TBMS.getCurrentEpisode(T_i,j, M_i,j, Length)
      Rules = TBMS.RetrieveRelatedEpisodes(T_i,j, M_i,j, Episode)
      For each rule in Rules
            Support = rule.getSupport()
            Confidence = rule.getConfidence()
            If (Support > suppThreshold and Confidence > confThreshold)
                  Results.add(rule)
                  Length = -1 // Break while loop for not keep
decreasing the length
      End For
      Length= Length - 1
End While

If (Results = Ø) // If there is no results in this level, we check the
lower level traces
      For each trace T_i-1,k in TBMS.getChilds(T_i,j, M_i,j)
            Results.add(Retrieve (T_i-1, k, M_i-1, k))
      End For
End If
Return Results
```

GetCurrentEpisode: Function that returns the last n obsels from the current trace $T_{i,j}$.
RetrieveRelatedEpisodes: Retrieves all the historical episodes in which obsels are matched with the same temporal order.
GetSupport, GetConfidence: Calculates support and confidence depending on **Definition 2, Definition 3**.
Add: a procedure for adding a rule with its support and confidence values to the results list. Rules are ordered from the highest to lowest rank, by using a ranking function depending on the support and confidence values.
GetChilds: Returns the child traces of a specific trace (move down a level).
Adapt: is a function that adapts the solution by taking into account the differences between the current problem and the retrieved episodes. The adaptation process is well explained in [6].

Sometimes, obsels order shouldn't be so strict, for example to do a specific task we may not care about which obsel should happen before another, but we care that both of them should happen before a third one. By considering this idea we can make an extension to **Algorithm 1,** to make it capable to accept episodes which don't match exactly the current episode. This extension could be done by using a loose array, with *n* columns and *n* rows, where *n* is the number of the obsel types, both of columns and rows are the obsel types, where each cell contains integer value indicating the loosing amount for changing the order between these two obsels. These values will be predefined according to each application. When the algorithm tries to retrieve the related episodes it will try all the possible changes of obsel orders and calculate the loosing value, if it was under a specific threshold, this try will be ignored.

As an example of use of this algorithm, we can consider that we have a user using SAP-BO Explorer. His objective is to visualize the quantity sold of all the products in New York in the year 2001, and then export his visualization into Excel file. Then he can send the results to his manager as a result of his exploration. For doing this, he starts by choosing the measure 'Quantity sold' and choosing the year and the city, and then he chooses the chart type. Now the TBA will try to retrieve the similar episodes in the stored traces, by trying to recognize if the current episode has a signature from the task signature base. TBA will recognize that the user is doing a task with the same signature in Figure 3, so TBA will retrieve all the episodes that have this signature and rank them. So it could suggest to the user either to export his data, or to send it by email. As the user is usually exporting the results more than sending it by email, so support and confidence values of export will be greater than send by email. That is why the export will be ranked first. So the user will choose to export his visualizing into Excel file.

4 Discussion

Our assistant is extensible and accepts any kind of traces; we can see that the proposed extension on **Algorithm 1** makes it sufficient because the matching is more flexible. This assistant is built over a TBMS, which means that the algorithm can use all TBMS functionalities. Thanks to task signatures extension, more tasks can be defined and more rules can be applied. By using these task signatures for the transformation, the user can understand what he is doing, by exploring the suggested episode and moving between the levels to see more details.

This paper is written in the context of a Master Thesis. We aim to develop a Trace-Based Assistant for SAP-BO Explorer, and try to make it as generic as we can. We are working to apply it to SAP-BO Explorer, to help business users make their analysis effectively, so we can support more experiments and examples to prove our approach. Some difficulties are facing us, because we need to have a huge amount of traces to be analysed, so we can build the task signature base and give the values to all the measures and thresholds that depend on the application.

Some of the issues and questions raised by the implementation in addition to defining some values are: How to adapt retrieved episodes? How to present the results to the user and be sure that he will be satisfied? What is the benefit of observing also the reaction of the user when the TBA suggests an episode to him? How to share the experience between the users? How to improve the accuracy of the ranking measures? All these questions are research issues that we intend to explore. In addition, we could consider as prospective some intelligent issues; the most important one is how to make the definition of task signatures automatically, because in our work we need system experts to define task signatures according to the observed application. We could also make the assistant able to understand the context of the work if the user opens many sessions of the same time. For example if the user opens two sessions to compare the sales between two cities in different periods, the assistant can recognize that the user is trying to do comparison, so a comparison table or chart could be generated automatically by the assistant in spite of the user.

5 Conclusion

In this paper, we have described an algorithm making use of interaction traces to provide an experience-based assistance to users of a specific application. This work is conducted in collaboration with SAP Business Object and the application we work on is called SAP-BO Explorer, a web application enabling users to load, explore, visualize and export data. The aim of the project is to develop a Trace-Based Assistant for this application, implementing state of the art results on this domain. The contribution of this paper focuses on one specific aspect of the assistant, the design of an experience retrieval algorithm. For that purpose, we have implemented a collect process in order to gather interaction traces, thus recording experiences of users in a reusable format. Then, with domain experts, we have defined "tasks signatures" to describe specific actions that users can perform on the application. Last, we have defined an algorithm able to retrieve in the stored interaction traces, previous episodes (i.e. previous users' experiences) matching a given task signature. This algorithm makes use of FSM in order to retrieve patterns corresponding to the signature. We have also shown that the retrieved episodes can be reused, and possibly adapted, in order to provide assistances to users.

At the time being, the collect process is implemented, but the retrieval algorithm is not finished yet. This is an immediate prospect for this work. Once this is done, we will be able to address other issues related to the development of a trace-based agent such as "how to adapt retrieved episodes to specific situations?".

6 References

[1] "SAP Business Objects Explorer: Explore your business at the speed of thought," *SAP*.

[2] P. Champin, Y. Prié, and A. Mille, "MUSETTE : a framework for Knowledge from Experience," *EGC'04, RNTI-E-2 (article court)*, Cepadues Edition, 2004, pp. 129–134.

[3] L.S. Settouti, Y. Prié, P. Champin, J. Marty, and A. Mille, *A Trace-Based Systems Framework : Models, Languages and Semantics*, 2009.

[4] D. Clauzel, K. Sehaba, and Y. Prié, "Modelling and Visualising Traces for Reflexivity in Synchronous Collaborative Systems," *2009 International Conference on Intelligent Networking and Collaborative Systems*, Barcelona, Spain: 2009, pp. 16-23.

[5] D. Cram, B. Fuchs, Y. Prié, and A. Mille, "An approach to User-Centric Context-Aware Assistance based on Interaction Traces," *MRC 2008, Modeling and Reasoning in Context*, 2008.

[6] A. Cordier, B. Mascret, and A. Mille, "Extending Case-Based Reasoning with Traces," *Grand Challenges for reasoning from experiences, Workshop at IJCAI'09*, 2009.

[7] S. Craw, "Agile case-based reasoning: A grand challenge towards opportunistic reasoning from experiences.," *IJCAI-09 Workshop on Grand Challenges in Reasoning from Experiences*, Pasadena, CA: 2009, pp. 33-39.

[8] R. Kanawati and M. Malek, "Behavioral Sequences: A New Log-Coding Scheme for Effective Prediction of Web User Accesses," *Adaptive Hypermedia and Adaptive Web-Based Systems*, 2006, pp. 406-410.

[9] F.J. Martin and E. Plaza, "Ceaseless Case-Based Reasoning," *Advances in Case-Based Reasoning*, 2004, pp. 43-62.

[10] F. Zehraoui, R. Kanawati, and S. Salotti, "CASEP2: Hybrid Case-Based Reasoning System for Sequence Processing," *Advances in Case-Based Reasoning*, 2004, pp. 127-132.

[11] M. Jaczynski and B. Trousse, "Broadway: A Case-Based System for Cooperative Information Browsing on the World-Wide-Web," *Collaboration between Human and Artificial Societies*, 1999, pp. 264-283.

[12] R. Agrawal and R. Srikant, "Mining sequential patterns," *Data Engineering, International Conference on*, Los Alamitos, CA, USA: IEEE Computer Society, 1995, p. 3.

[13] Q. Zhao and S.S. Bhowmick, *Sequential pattern mining: a survey*, Technical Report Center for Advanced Information Systems, School of Computer Engineering, Nanyang Technological University, Singapore, 2003.

[14] Y. Chen and Y. Hu, "Constraint-based sequential pattern mining: The consideration of recency and compactness," *Decision Support Systems*, vol. 42, Nov. 2006, pp. 1203-1215.

[15] "Using the Office Assistant," *Microsoft Corporation*.

[16] J. Laflaquière, L. Settouti, Y. Prié, and A. Mille, "Trace-Based Framework for Experience Management and Engineering," *Knowledge-Based Intelligent Information and Engineering Systems*, 2006, pp. 1171-1178.

[17] J. Carroll and D. Long, *Theory of finite automata with an introduction to formal languages*, Prentice-Hall, Inc., 1989.

[18] F. Wagner, R. Schmuki, and T. Wagner, *Modeling software with finite state machines*, CRC Press, 2006.